Diplomarbeit

im Studiengang Informatik

„Ein Hochschulschriften-Server für die SLUB Dresden - Weboberfläche für Browsing und Recherche“

vorgelegt von

Sebastian Schulz

geb. am 07.05.1976 in Frankfurt(Oder)
Matrikelnummer 2417475

angefertigt an der

Professur für Datenbanken
Institut für Systemarchitektur
Fakultät Informatik
Technische Universität Dresden

betreuender Hochschullehrer:
Prof. Dr. Klaus Meyer-Wegener

betreuende Mitarbeiter:
Dr. Bernd Keller (TU Dresden)
Dagmar Wohlfarth (SLUB Dresden)
Jürgen Grothe (SLUB Dresden)

Beginn der Arbeit: 01.11.2000
Abgabe der Arbeit: 31.07.2001

Inhaltsverzeichnis

Inhaltsverzeichnis 3

Abbildungsverzeichnis 9

Tabellenverzeichnis 13

Listings 17

Selbständigkeitserklärung 19

1. Einführung 21

1.1. Motivation 22

1.2. Inhaltsüberblick 24

2. Aufgabenstellung, Grundlagen, Arbeitsabläufe 27

2.1. Aufgabenstellung der Diplomarbeit 28

2.2. Interpretation der Aufgabenstellung 30

2.3. Rückblick auf den Großen Beleg 33

2.3.1. Anforderungen und Ergebnisse der Belegarbeit 33

2.3.2. Verbesserungsmöglichkeiten 34

2.4. Digitale Archive und elektronisches Publizieren im bibliothekarischen Umfeld 36

2.4.1. Open Archives Initiative (OAI) 38

2.4.1.1. Protocol for Metadata Harvesting 40

2.4.2. Deutsche Initiative für Netzwerkinformationen (DINI) 44

2.4.2.1. Ziele der Arbeitsgruppe „Elektronisches Publizieren“ . 45

2.4.2.2. Anforderungen an einen Hochschulschriftenserver . . . 46
2.4.2.3. Einschätzung und Wertung . . . 50
2.4.3. Dissertationen Online . . . 50
2.4.3.1. Dissertation Markup Language (DiML) . . . 51
2.4.4. Subjektive Einschätzung der betrachteten Initiativen und Entwicklungen . . . 55
2.4.5. Realisierungsmöglichkeiten für den HSSS . . . 58

3. Systemarchitektur 61
3.1. Geschäftsmodelle und Arbeitsabläufe . . . 62
3.1.1. Geschäftsmodell für Dissertationen . . . 64
3.1.2. Arbeitsablauf für Berichte . . . 71
3.1.3. Einschätzung der Arbeitsabläufe . . . 73
3.2. Architekturübersicht . . . 78
3.2.1. Architekturvision . . . 80
3.2.2. Vorgehensweise, Priorisierung . . . 88
3.3. Architekturdetails . . . 91
3.3.1. Architekturbesonderheiten . . . 92
3.3.1.1. Bearbeitungsnummer . . . 92
3.3.1.2. Konfigurationsstrukturen . . . 94
3.3.2. Architekturdetails auf Server-Ebene . . . 96
3.3.2.1. Technologische Alternativen . . . 97
3.3.2.2. Objektorientierte Analyse und Design . . . 99
3.3.2.3. Architektur- und Entwurfsmuster . . . 105
3.3.3. Architekturdetails auf Client-Ebene . . . 108
3.3.3.1. Einordnung in die Gesamtarchitektur . . . 108
3.3.3.2. Technologische Alternativen . . . 111
3.3.4. Architekturdetails auf Datenbank-Ebene . . . 113
3.3.4.1. Schema- und Datenbankanforderungen . . . 114
3.3.4.2. Datenbank-Evaluation . . . 115
3.3.4.3. Design der Schnittstelle . . . 118

4. Teilanwendung 1 : MetadatenGenerator **121**

4.1. Objektorientierte Analyse und Design 122
4.2. Realisierung 129
4.3. Persistenzrealisierung 134
4.4. Zusätzliche Funktionalitäten 136
4.4.1. Evaluation 136
4.4.2. Status-Verfolgung durch den Autor 137
4.4.3. MetadatenGenerator Light - eine Alternativ-Version 139
4.5. Einschätzung 142

5. Teilanwendung 2 : AdminTools **145**

5.1. Analyse und Design 146
5.2. Realisierung 151
5.2.1. Unterstützung der Arbeitsabläufe 151
5.2.2. Administrationsunterstützung 162
5.3. Persistenzrealisierung 166
5.4. Einschätzung 174

6. Teilanwendung 3 : HSSS AccessTools / myHSSS **177**

6.1. Realisierung einer absoluten URL 178
6.2. Objektorientierte Analyse und Design 185
6.2.1. Die Volltextsuchmaschine Lucene 186
6.2.2. Integration von Lucene in die Anwendung 191
6.3. Realisierung 197
6.4. Einschätzung 205

7. Implementierung des OAI-Protokolls **209**

7.1. Objektorientierte Analyse und Design 210
7.2. Realisierung 215
7.3. Einschätzung 218

8. Zusammenfassung und Ausblick **219**

8.1. Zusammenfassung 220
8.2. Verbesserungsmöglichkeiten 222
8.2.1. Verbesserungsmöglichkeiten des Gesamtsystems 222
8.2.2. Verbesserungsmöglichkeiten für den MetadatenGenerator 224
8.2.3. Verbesserungsmöglichkeiten für die AdminTools 225
8.2.4. Verbesserungsmöglichkeiten für die AccessTool / myHSSS . . . 226
8.3. Ausblick 227

A. Metadatensätze des HSSS **229**
A.1. Beschreibungkriterien 230
A.2. Spezifikation der Metadaten 231
A.2.1. DC.TITLE 231
A.2.2. DC.CREATOR 232
A.2.3. DC.SUBJECT 233
A.2.4. DC.DESCRIPTION 235
A.2.5. DC.PUBLISHER 236
A.2.6. DC.CONTRIBUTOR 237
A.2.7. DC.DATE 238
A.2.8. DC.TYPE 239
A.2.9. DC.IDENTIFIER 239
A.2.10. DC.FORMAT 240
A.2.11. DC.SOURCE 241
A.2.12. DC.LANGUAGE 242
A.2.13. DC.RELATION 242
A.2.14. DC.RIGHTS 243
A.3. Spezifikation der temporären Metadaten 244

B. Konfigurationsstrukturen des HSSS **249**
B.1. Überblick 250
B.2. web.xml 251
B.3. hsssdata.xml 253

B.4. MetadataSetDissertation.xml / MetadataSetBericht.xml 255
B.5. oaisets.xml 257

C. Datenbank-Schema des HSSS **259**
C.1. Überblick 260
C.2. Schema 265

D. Verwendete Technologien / Produkte **275**
D.1. Kommerzielle Technologien / Produkte 276
D.1.1. Sybase 276
D.2. Open Source - Technologien / - Produkte 277
D.2.1. JSDK 277
D.2.2. Apache 277
D.2.3. Tomcat 278
D.2.4. Lucene 278
D.2.5. Xerces 279
D.2.6. Xalan 279
D.2.7. FOP 280
D.2.8. JDOM 280
D.2.9. PoolMan 281
D.2.10. PJ 281
D.2.11. FINJ 282
D.2.12. JavaMail 282

E. Literaturverzeichnis und Linkverzeichnis **285**
Literaturverzeichnis 286
Linkverzeichnis 290

Abbildungsverzeichnis

Abbildung 1 - schematische Darstellung der HSSS-Gesamtanwendung . . . 31

Abbildung 2 - Anwendungsfalldiagramm:
Kontext des Systems für den Autor 64

Abbildung 3 - Sequenzdiagramm: Geschäftsgang in Bezug auf den Autor . 65

Abbildung 4 - Anwendungsfalldiagramm:
Kontext des Systems für das Referat 5.3 66

Abbildung 5 - Sequenzdiagramm:
Geschäftsgang in Bezug auf das Referat 5.3 - Teil 1 67

Abbildung 6 - Sequenzdiagramm:
Geschäftsgang in Bezug auf das Referat 5.3 - Teil 2 69

Abbildung 7 - Anwendungsfalldiagramm:
Kontext des Systems für das Referat 3.4 70

Abbildung 8 - Sequenzdiagramm:
Geschäftsgang in Bezug auf das Referat 3.4 70

Abbildung 9 - Sequenzdiagramm:
Geschäftsgang für Berichte - Teil 1 72

Abbildung 10 - Sequenzdiagramm:
Geschäftsgang für Berichte - Teil 2 73

Abbildung 11 - Doppelte Titelaufnahme einer Ressource 75

Abbildung 12 - Einheitliche Titelaufnahme einer Ressource 76

Abbildung 13 - Trennung von Inhalt / Repräsentation 76

Abbildung 14 - Komponentenmodell der HSSS-Gesamtanwendung 78

Abbildung 15 - Systemkomponenten im Schichtenmodell 79

Abbildung 16 - Abbildung Objektstruktur - XML-Struktur 81

Abbildung 17 - Einfluss der Metadatensatzbeschreibung auf die Anwendung 83
Abbildung 18 - Drei-Schichten-Modell 91
Abbildung 19 - An der Gesamtanwendung beteiligte Komponenten 96
Abbildung 20 - HSSS Package-Struktur 100
Abbildung 21 - HSSS Verzeichnishierarchie 101
Abbildung 22 - HSSS Package “persistence“ (Ausschnitt) 102
Abbildung 23 - HSSS Package “utilities“ (Ausschnitt) 103
Abbildung 24 - HSSS Package “xml“ (Ausschnitt) 104
Abbildung 25 - Klassendiagramm „Singleton“ 106
Abbildung 26 - Klassendiagramm „Fabrikmethode“ 106
Abbildung 27 - HSSS Package “temporalpersistence“ (Ausschnitt) 107
Abbildung 28 - Architektur „page centric approach“ (2-Tier) 108
Abbildung 29 - Architektur „dispatcher approach“ (MVC) 109
Abbildung 30 - HSSS Package “db“ 118

Abbildung 31 - HSSS Package ”metadatagenerator” 122
Abbildung 32 - View-Controller-Aktivitäten des MetadatenGenerators . . 124
Abbildung 33 - HSSS Package ”metadataset” (Ausschnitt) 126
Abbildung 34 - SLUB MetadatenGenerator: Formular 1 129
Abbildung 35 - SLUB MetadatenGenerator: Hilfe 130
Abbildung 36 - SLUB MetadatenGenerator: Formular 2 131
Abbildung 37 - SLUB MetadatenGenerator: Übersicht 132
Abbildung 38 - SLUB MetadatenGenerator: Fragebogen (Ausschnitt) . . . 137
Abbildung 39 - SLUB MetadatenGenerator: Bearbeitungsstatus 138
Abbildung 40 - SLUB MetadatenGenerator Light (Ausschnitt) 140

Abbildung 41 - HSSS AdminTools: Übersicht der Metadatensätze 152
Abbildung 42 - HSSS AdminTools: Die Metadaten eines Metadatensatzes . 154
Abbildung 43 - HSSS AdminTools: Druck-Präsentation der Primär-Metadaten 155
Abbildung 44 - HSSS AdminTools: Export nach DDB 156
Abbildung 45 - HSSS AdminTools: temporäre Metadaten, Zustände 158

Abbildung 46 - HSSS AdminTools: Einstellen des Metadatensatzes 159
Abbildung 47 - HSSS AdminTools: Benutzerverwaltung 163
Abbildung 48 - HSSS AdminTools: Konfigurationsstrukturen 164
Abbildung 49 - Grundlegende Schemastruktur 1 167
Abbildung 50 - Grundlegende Schemastruktur 2 168
Abbildung 51 - Grundlegende Schemastruktur 3 169

Abbildung 52 - Vorgang der URL-Abbildung 182
Abbildung 53 - HSSS Frontdoor 184
Abbildung 54 - Lucene Index-Merge 189
Abbildung 55 - HSSS Package "lucene" 191
Abbildung 56 - myHSSS: Volltextsuche in der Auflösung 800*600 197
Abbildung 57 - myHSSS: Feldsuche in der Auflösung 1024*768 198
Abbildung 58 - myHSSS: Feldsuche in der Einzel-Ansicht 199
Abbildung 59 - myHSSS: Hilfe in der Einzel-Ansicht 200
Abbildung 60 - myHSSS: gesammelte Treffer 201
Abbildung 61 - myHSSS: Teilergebnis einer Volltextsuche 202

Abbildung 62 - HSSS Package "oai" 210
Abbildung 63 - XMl-Struktur der Antwort auf eine „Identify"-Anfrage . . 215
Abbildung 64 - OAI Repository Explorer: Anzeige eines Records 216
Abbildung 65 - OAI-Informationsseite des HSSS 217

Tabellenverzeichnis

Tabelle 1 - Lebensdauer verschiedener Datenträger 37

Tabelle 2 - Am Geschäftsmodell beteiligte Abteilungen der SLUB 63
Tabelle 3 - Einflussmöglichkeiten der Metadatensatzbestandteile 84
Tabelle 4 - Konfigurationsstrukturen des HSSS 95

Tabelle 5 - die Tabelle `temp_metadatasets` 134

Tabelle 6 - Dokument-spezifische Zustände 147
Tabelle 7 - Metadaten-spezifische Zustände 147
Tabelle 8 - Gesamtzustände und Abhängigkeiten 148
Tabelle 9 - Bearbeitungsmöglichkeiten von Metadatensätzen 153
Tabelle 10 - Zuordnung der DC-Kategorien zu den Strukturen 170

Tabelle 11 - Vor- und Nachteile der Abbildungsansätze 180
Tabelle 12 - Dokumentenfelder und Inhalte 192

Tabelle A1 - Kriterien für die Metadatenbeschreibung 230
Tabelle A2 - HSSS-Metadatensatz : Originaltitel der Documents 231
Tabelle A3 - HSSS-Metadatensatz : Untertitel / weitere Titel 231
Tabelle A4 - HSSS-Metadatensatz : Übersetzter Titel 232
Tabelle A5 - HSSS-Metadatensatz : Name des Verfassers 232
Tabelle A6 - HSSS-Metadatensatz : Geburtsdatum des Verfassers 232
Tabelle A7 - HSSS-Metadatensatz : Geburtsort des Verfassers 233
Tabelle A9 - HSSS-Metadatensatz : Adresse des Verfassers 233
Tabelle A10 - HSSS-Metadatensatz : Eigene Schlagworte (deutsch) 233

Tabelle A11 - HSSS-Metadatensatz : Eigene Schlagworte (englisch) 234
Tabelle A12 - HSSS-Metadatensatz : DNB-Klassifikation 234
Tabelle A14 - HSSS-Metadatensatz : RVK-Klassifikation 234
Tabelle A15 - HSSS-Metadatensatz : SWD-Schlagworte 235
Tabelle A16 - HSSS-Metadatensatz : Beschreibung (deutsch) 235
Tabelle A17 - HSSS-Metadatensatz : Beschreibung (englisch) 235
Tabelle A18 - HSSS-Metadatensatz : Name der Universitätsbibliothek . . . 236
Tabelle A19 - HSSS-Metadatensatz : Adresse der Universitätsbibliothek . . 236
Tabelle A20 - HSSS-Metadatensatz : Name der universitären Einrichtung . 237
Tabelle A22 - HSSS-Metadatensatz : Adresse der universitären Einrichtung 237
Tabelle A23 - HSSS-Metadatensatz : Gutachter 237
Tabelle A24 - HSSS-Metadatensatz : Betreuer 238
Tabelle A26 - HSSS-Metadatensatz : Tag der mündlichen Prüfung 238
Tabelle A27 - HSSS-Metadatensatz : Tag der Abgabe 238
Tabelle A28 - HSSS-Metadatensatz : Tag der Abgabe (SLUB) 239
Tabelle A29 - HSSS-Metadatensatz : Dokumenten-Typ 239
Tabelle A30 - HSSS-Metadatensatz : stabile Dokument-URL 239
Tabelle A31 - HSSS-Metadatensatz : reale Dokument-URL 240
Tabelle A32 - HSSS-Metadatensatz : HSSS-Format 240
Tabelle A33 - HSSS-Metadatensatz : Abgabe-Formate 240
Tabelle A34 - HSSS-Metadatensatz : Original-Formate 241
Tabelle A35 - HSSS-Metadatensatz : Veröffentlichungs-Titel 241
Tabelle A36 - HSSS-Metadatensatz : Veröffentlichungs-ISBN 241
Tabelle A37 - HSSS-Metadatensatz : Sprache des Dokuments 242
Tabelle A38 - HSSS-Metadatensatz : Literaturquellen (extern) 242
Tabelle A39 - HSSS-Metadatensatz : Literaturquellen (intern) 242
Tabelle A40 - HSSS-Metadatensatz : rechtliche Bedingungen 243
Tabelle A41 - HSSS-Metadatensatz : Dokumentübermittlung 244
Tabelle A42 - HSSS-Metadatensatz : Eingangsdatum 244
Tabelle A43 - HSSS-Metadatensatz : Abgabedatum des Dokuments 244

Tabelle A44 - HSSS-Metadatensatz : Datum der letzten Bearbeitung . . . 245
Tabelle A45 - HSSS-Metadatensatz : Bearbeitungsnummer 245
Tabelle A46 - HSSS-Metadatensatz : Gesamt-Bearbeitungsstatus 245
Tabelle A47 - HSSS-Metadatensatz : Dokument-Bearbeitungsstatus 246
Tabelle A48 - HSSS-Metadatensatz : Metadaten-Bearbeitungsstatus 246
Tabelle A49 - HSSS-Metadatensatz : Exportinformation DDB 246
Tabelle A50 - HSSS-Metadatensatz : Exportinformation SWB 247
Tabelle A51 - HSSS-Metadatensatz : Dokumentinformationen 247
Tabelle A52 - HSSS-Metadatensatz : E-Mail Adresse des Autors 247
Tabelle A53 - HSSS-Metadatensatz : Telefonnummer des Autors 248
Tabelle A54 - HSSS-Metadatensatz : Nachricht des Autors 248

Tabelle A55 - Bedeutung der Relationen-Prefixe 261
Tabelle A56 - Tabellenübersicht (Teil 1) 261
Tabelle A57 - Tabellenübersicht (Teil 2) 262
Tabelle A58 - Übersicht der gespeicherten Prozeduren (Teil 1) 263
Tabelle A59 - Übersicht der gespeicherten Prozeduren (Teil 2) 264
Tabelle A60 - die Tabelle `config_hsss` 265
Tabelle A61 - die Tabelle `config_metadatasets` 265
Tabelle A62 - die Tabelle `stat_metadatasets` 265
Tabelle A63 - die Tabelle `temp_metadatasets` 266
Tabelle A64 - die Tabelle `stat_metadatasets_finished` 266
Tabelle A65 - die Tabelle `pers_address` 267
Tabelle A66 - die Tabelle `pers_date` 267
Tabelle A67 - die Tabelle `pers_description` 267
Tabelle A68 - die Tabelle `pers_format` 267
Tabelle A69 - die Tabelle `pers_kindofdocument` 268
Tabelle A70 - die Tabelle `pers_language` 268
Tabelle A71 - die Tabelle `pers_person` 268
Tabelle A72 - die Tabelle `pers_publication` 268
Tabelle A73 - die Tabelle `pers_relation` 269

Tabelle A74 - die Tabelle `pers_rights` 269
Tabelle A75 - die Tabelle `pers_subject` 269
Tabelle A76 - die Tabelle `pers_contributor` 269
Tabelle A77 - die Tabelle `pers_publisher` 270
Tabelle A78 - die Tabelle `pers_type` 270
Tabelle A79 - die Tabelle `pers_title` 270
Tabelle A80 - die Tabelle `pers_metadataset` 271
Tabelle A81 - die Tabelle `pers_fulltext` 271
Tabelle A82 - die Tabelle `pers_date_metadataset` 272
Tabelle A83 - die Tabelle `pers_description_metadataset` 272
Tabelle A84 - die Tabelle `pers_format_metadataset` 272
Tabelle A85 - die Tabelle `pers_person_metadataset` 273
Tabelle A86 - die Tabelle `pers_publication_metadataset` 273
Tabelle A87 - die Tabelle `pers_relation_metadataset` 273
Tabelle A88 - die Tabelle `pers_subject_metadataset` 274
Tabelle A89 - die Tabelle `pers_title_metadataset` 274

Listings

Listing 1 - *Record-Struktur des OAI-Protokolls* 41
Listing 2 - *Struktur einer OAI-Protokoll-Antwort* 42
Listing 3 - *Struktur eines Deckblattes in DiML* 53
Listing 4 - *formale XML-Struktur eines Metadatensatzes* 82
Listing 5 - *formale XML-Struktur einer Metadatensatzbeschreibung* 85
Listing 6 - *grundlegende Struktur der Konfigurationsdatei „web.xml“* . . . 95
Listing 7 - *Einfaches Beispiel für den Einsatz von TagLibs* 110
Listing 8 - *Export-Bereich der Konfigurationsstruktur „hsssdata.xml“* . . . 149
Listing 9 - *stored procedure* 171
Listing 10 - *Realisierung des „Pagings“* 195
Listing A1 - *formale XML-Struktur eines Metadatensatzes* 256
Listing A2 - *XML-Struktur der OAI-Sets* 257

Selbständigkeitserklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.
Ich bin damit einverstanden, dass die Arbeit veröffentlicht wird und dass in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird. Der Technischen Universität Dresden, vertreten durch die Professur für Datenbanken, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Dresden, den 31. Juli 2001

1. Einführung

Inhalt des Kapitels:

1.1. Motivation **22**
1.2. Inhaltsüberblick **24**

1.1. Motivation

Auch an den Bibliotheken und insbesondere an den Universitätsbibliotheken geht das digitale Zeitalter nicht spurlos vorbei. In Deutschland stehen vor allem die Universitätsbibliotheken zu Beginn des 21. Jahrhunderts vor der großen Herausforderung, die sich rasch vollziehenden Veränderungen und die sich bietenden technischen Möglichkeiten zu erkennen und als Chance zu begreifen, sich vom Image angestaubter Archivieranstalten zu lösen und sich nach und nach zu „universitären Informations- und Servicezentren" zu entwickeln [Sie00a][Wis01].

Dabei kann sich die gegenwärtig durch verschiedene Umstände zunehmend eskalierende Finanzsituation vieler Bibliotheken durchaus als positives Moment einer Veränderung auswirken, weil sie den Reformdruck enorm erhöht. Auch große wissenschaftliche Verlage im kommerziellen Bereich stehen weltweit vor der Aufgabe, ihre Geschäftsfelder neu auszurichten und zunehmend auf Mehrwertdienste außerhalb des Vertriebs von wissenschaftlichen Publikationen auf herkömmlichem Wege (Zeitschriften etc.) zu orientieren, so etwa auf Information-Retrieval- oder Publishing-on-Demand-Systeme.

Diesen Anforderungen werden sich auch die Bibliotheken stellen müssen. Zu ihren wichtigsten Aufgaben wird schon in naher Zukunft vor allem die Bereitstellung neuer wissenschaftlicher Erkenntnisse in elektronischer und leicht recherchierbarer Form gehören. Dazu ist der Betrieb professioneller Dokumentenserver und Suchmaschinen notwendig, aber auch die organisatorische Bewältigung von Begutachtung und Zertifizierung der Veröffentlichungen.

Für die Politik (insbesondere das Bundesforschungsministerium) kommt es darauf an, die Bibliotheken bei der großen Herausforderung einer organisierten, flächendeckenden Realisierung dieser Infrastrukturen so effektiv und vor allem so schnell wie möglich zu unterstützen. Die Meinung, dass das Internet die Universitätsbibliotheken überflüssig mache, ist schlichtweg unsinnig. Richtig aber ist, dass sich die Bibliothekare zunehmend von „Sammlern zu Jägern", also von „Archivaren zu Informationbrokern" entwickeln müssen [Sie00a]. Noch mehr muss sich die Erkenntnis durchsetzen, dass es sich bei den wissenschaftlichen Publikationen um reale Werte handelt, die nicht zuletzt auch durch staatliche Fördermittel entstanden sind. Wirklicher Nutzen kann aus ihnen aber nur entstehen, wenn sie der Allgemeinheit zugänglich gemacht werden. Das aber geschieht nicht durch die bloße Existenz des Internets.

Auch für die Sächsische Landesbibliothek - Staats- und Universitätsbibliothek (SLUB - im Folgenden nur noch SLUB genannt) trifft diese Standortbestimmung zu. Hier machte man sich etwa ab dem Jahr 1999 verstärkt darüber Gedanken, wie man in Zukunft Hochschulschriften digital verwalten und archivieren könnte. Dabei orientierte man sich an Universitätsbibliotheken bzw. Universitäten, die zu diesem Zeitpunkt bereits einen Dienst für digitale Hochschulschriften anbieten konnten[1]. Dadurch konnten erste Erkenntnisse gewonnen werden, welche Anforderungen ein solcher Hochschulschriftenserver abdecken sollte.

Ab Dezember 1999 wurden diese Erkenntnisse in Zusammenarbeit mit der Professur Datenbanken der Fakultät Informatik der TU Dresden im Rahmen einer Belegarbeit erheblich vertieft[Schb00]. Zusätzlich konnten Teile eines speziell auf die Belange der SLUB zugeschnittenen Hochschulschriftenservers (im Folgenden nur noch HSSS genannt) unter Nutzung moderner Technologien prototypisch implementiert und erfolgreich getestet werden. Dabei handelte es sich um Web-basierte Anwendungen, die die Eingabe von Metadaten durch die Autoren der Hochschulschriften unterstützten bzw. die Auswertung und Validierung derselben durch Mitarbeiter der SLUB.

Im Zeitraum zwischen Juni und November 2000 wurden die Ergebnisse der Belegarbeit von Mitarbeitern der SLUB ausgewertet und zusätzliche Anforderungen definiert. Im November 2000 erfolgte mit der Aushändigung dieses Diplomthemas der Startschuss zur Weiterentwicklung des HSSS, aufbauend auf den Ergebnissen der Belegarbeit (der genaue Wortlaut des Diplomthemas ist im Abschnitt 2.1 auf Seite 28 zu finden). Ziel der Diplomarbeit ist es nicht, die endgültige Produktiv-Version eines Hochschulschriftenservers für die SLUB zu erstellen[2], sondern einen möglichst vollständigen Umfang der Dienste des HSSS in einer ersten Ausbaustufe prototypisch zu implementieren. Das spezielle Augenmerk liegt dabei auf tieferen Architektur- und Arbeitsablauferkenntnissen, die erst durch eine solche prototypische Realisierung gewonnen werden können und die für die SLUB von „universeller“ Natur sind, unabhängig davon, wie sich die weitere Entwicklung des HSSS gestalten wird.

[1] vor allem OPUS der Universität Stuttgart

[2] was nach Auffassung des Autors auch gar nicht sinnvoll und möglich wäre

1.2. Inhaltsüberblick

Zunächst wurde als Einführung und Motivation die Ausgangssituation beschrieben, in der sich die meisten Universitätsbibliotheken im Bezug auf elektronische Dokumentenverwaltung befinden. Danach ist diese Situation für die SLUB konkretisiert und damit speziell die Motivation für Thema und Gegenstand der Diplomarbeit abgeleitet worden. Es folgte ein kurzer Abriss der Bemühungen, die zu diesem Thema von Seiten der SLUB bereits unternommen worden sind.

Im sich anschließenden Kapitel wird die Aufgabenstellung dieser Diplomarbeit vorgestellt und kurz interpretiert. Danach folgt ein Rückblick auf die Belegarbeit, die der Diplomarbeit inhaltlich vorausging. Es werden Anforderungen und Ergebnisse der Belegarbeit erläutert, Stärken und Schwächen analysiert und damit auch Konsequenzen für die Diplomarbeit gezogen. Anschließend wird noch einmal auf die Situation der Universitätsbibliotheken und deren Umfeld eingegangen, insbesondere auf technische Schwierigkeiten und Probleme, aber auch auf interessante Möglichkeiten, neue Protokolle, Schnittstellen und Initiativen. Anhand von drei Initiativen bzw. Organisationen, die etwas detaillierter und durchaus auch kritisch betrachtet werden, soll die Entwicklung auf diesem Gebiet in den letzten Jahren nachvollzogen und eingeschätzt sowie Anknüpfungspunkte für die eigene Entwicklung aufgezeigt werden.

Anschließend soll die Architektur des eigenen Systems vorgestellt werden, beginnend mit der Analyse und Spezifikation der Arbeitsabläufe bzw. Geschäftsgänge, die organisatorisch zu etablieren und von System weitestgehend zu unterstützen sind. Danach werden zunächst Architekturvisionen beschrieben, die sich auf das System in seiner Gesamtheit beziehen und eine Vorstellung davon liefern sollen, welche Schwerpunkte und Besonderheiten der Entwurf möglichst berücksichtigen soll. Diese Architekturversion werden daraufhin auf ihre Praxistauglichkeit überprüft und auf die in der Diplomarbeit realisierbaren Teile begrenzt. Anschließend werden Architekturdetails der Gesamtarchitektur aus verschiedenen Blickwinkeln beleuchtet, so etwa der Aufbau von der für die Anwendungen verwendeten Konfigurationsstrukturen und die Gesamtarchitektur der Client-, Server- und Datenbankebene.

Nachdem die Grundlagen des Entwurfs auf der Abstraktionsebene der Gesamtarchitektur betrachtet wurde, wendet sich die Diplomarbeit den einzelnen Teilanwendungen zu. Zunächst wird die Teilanwendung MetadatenGenerator betrachtet, welche den Autoren von Hochschulschriften die Erstellung und Übermittlung entsprechender Me-

tadaten ermöglichen soll. Es werden zuerst spezifische Designentscheidungen erläutert, bevor die Realisierung der Teilanwendung beschrieben wird. Danach wird die für diese Teilanwendung verwendete Persistenzrealisierung betrachtet. Ein weiterer Abschnitt beschreibt einige zusätzliche Funktionalitäten, die während der Diplomarbeit in Zusammenarbeit mit zwei Auszubildenden der SLUB im Bereich dieser Teilanwendung entstanden sind. Schließlich werden die erzielten Ergebnisse eingeschätzt und mögliche Verbesserungen respektive Erweiterungen vorgestellt.

Die Teilanwendung *AdminTools* wird im nächsten Kapitel beschrieben, wobei zunächst wieder Besonderheiten der Teilanwendung in Bezug auf deren Design bzw. deren Architektur betrachtet werden. Anschließend wird die Realisierung dieser sehr umfangreichen Anwendung detailliert beschrieben, wobei die beiden Funktionsbereiche Arbeitsablaufunterstützung und Administrationsunterstützung unterschieden werden. Danach wird im Rahmen der Persistenzrealisierung dieser Teilanwendung der wesentliche Teil des Datenbank-Schemas uns dessen Design beschrieben. Am Ende erfolgt wieder eine Einschätzung der Implementation sowie ein Ausblick auf mögliche Verbesserungen bzw. Erweiterungen.

Im dem sich anschließenden Kapitel wird die dritte Teilanwendung betrachtet, welche den Web-basierten Zugriff auf die im Gesamtsystem vorhandenen Hochschulschriften bzw. deren Metadaten realisieren soll. Zunächst wird jedoch eine Indirektion betrachtet, die es erlaubt, stabile URL's zu garantieren. Diese sind eine wichtige Voraussetzung für die langfristige Verfügbarkeit der Hochschulschriften. Anschließend wird das Design der dritten Teilanwendung näher erläutert, wobei der Fokus insbesondere auf der Beschreibung und der Integration einer externen Volltextsuchmaschine liegt. Danach wird die entstandene Realisierung vor allem im Bezug zu den für diese Teilanwendung aufgestellten Architekturvisionen betrachtet, bevor wieder eine Einschätzung der Ergebnisse das Kapitel abschließt.

Danach erfolgt eine Erläuterung der Implementation des OAI-Protokolls, die es erlaubt, dass der HSSS gegenüber anderen OAI-Archiven als *Data Provider* auftreten kann. Hier werden vor allem die Besonderheiten der Realisierung betrachtet, die über eine Minimalumsetzung des Protokolls hinausgehen.

Im letzten Kapitel erfolgt eine Zusammenfassung und eine Einschätzung der Ergebnisse, die im Laufe der Diplomarbeit realisiert werden konnten. Dabei bezieht sich die Einschätzung auf die Kapitel 2 bzw. 3 aufgestellten Anforderungen respektive Architekturvisionen. In einem weiteren Abschnitt dieses Kapitels werden mögliche Verbesserungen bzw. Erweiterungen des Gesamtsystems in einer strukturierten Art und

Weise vorgestellt. Abschließend erfolgt ein Ausblick, der die weiteren Entwicklungs- und Einsatzmöglichkeiten sowie die Prämissen vorstellt, die aus Sicht des Autors notwendig sind, um die bisher erzielten Ergebnisse in eine stabile Produktivversion zu überführen.

Im Anhang der Diplomarbeit finden sich Informationen zu den Metadatensatzspezifikationen, zu den Konfigurationsstrukturen sowie zum Datenbankschema des HSSS. Außerdem ist eine kurze Auflistung der wichtigsten externen API's bzw. Technologien enthalten, die in das Gesamtsystem des HSSS integriert wurden.

2. Aufgabenstellung, Grundlagen, Arbeitsabläufe

Inhalt des Kapitels:

2.1. Aufgabenstellung der Diplomarbeit 28
2.2. Interpretation der Aufgabenstellung 30
2.3. Rückblick auf den Großen Beleg 33
2.3.1. Anforderungen und Ergebnisse der Belegarbeit 33
2.3.2. Verbesserungsmöglichkeiten 34
2.4. Digitale Archive und elektronisches Publizieren im bibliothekarischen Umfeld 36
2.4.1. Open Archives Initiative (OAI) 38
2.4.1.1. Protocol for Metadata Harvesting 40
2.4.2. Deutsche Initiative für Netzwerkinformationen (DINI) 44
2.4.2.1. Ziele der Arbeitsgruppe „Elektronisches Publizieren“ 45
2.4.2.2. Anforderungen an einen Hochschulschriftenserver . 46
2.4.2.3. Einschätzung und Wertung 50
2.4.3. Dissertationen Online 50
2.4.3.1. Dissertation Markup Language (DiML) 51
2.4.4. Subjektive Einschätzung der betrachteten Initiativen und Entwicklungen 55
2.4.5. Realisierungsmöglichkeiten für den HSSS 58

2.1. Aufgabenstellung der Diplomarbeit

Der Autor dieser Diplomarbeit hat in einem Großen Beleg, der dieser Arbeit inhaltlich vorausging, ein System für die Übergabe von Hochschulschriften an die SLUB erstellt. Auf der Basis dieses Systems ist nun ein neuer Baustein zu konzipieren und zu erstellen, der auch die Nutzung der Schriften über das Web ermöglicht. Dazu ist der zweite Teil der schon vorliegenden Anwendung, in der die SLUB die von den Autoren übergebenen Daten prüft und sie um eigenen Verwaltungsdaten ergänzt, noch einmal zu überarbeiten und an die neuen Aufgaben anzupassen. Insbesondere soll der Einsatz eines Datenbanksystems für die Metadaten erfolgen. Es ist ein geeignetes Datenbankverwaltungssystem auszuwählen, das sich möglichst gut in die verwendete Entwicklungsumgebung einpasst. Die bisherige dateibasierte Lösung muss abgelöst werden, da sie für den gleichzeitigen Zugriff verschiedener Benutzer (Autor, SLUB, Leser) und die Suche nicht ausreichend Unterstützung bietet.

Damit entsteht die Notwendigkeit, ein Datenbankschema zu entwerfen. Hier sind auch die Möglichkeiten von XML zu prüfen. XML könnte die Generierung der verschiedenen Sichten auf die Daten und die unterschiedlichen Layouts sehr unterstützen. Die Ablage der Dokumente ist dabei auch in dieser Arbeit noch keine Aufgabe für die Datenbank; sie erfolgt weiter in Dateien, auf die aus der Datenbank heraus verwiesen wird. Dabei ist der Dateiname vollständig zu verbergen; weder der Autor noch der Leser sollen ihn jemals erfahren. Statt dessen erhalten sie eine stabile URL zu der Schrift, die sich über Jahre hinweg nicht ändern soll - auch wenn die Dateistrukturen intern geändert werden. Es ist deshalb eine Indirektion zu realisieren (ggf. unter Einbeziehung der Datenbank), die die URL auf den Dateinamen abbildet. Genauer sollte diese URL zunächst auf eine sog. „Frontdoor“ der Schrift führen, also eine HTML-Seite, die alle Angabe zu dieser Schrift enthält und auch einen Link zum Dokument selbst, evtl. in mehreren Formaten (PDF, PostScript, HTML, später vielleicht XML). Nachdem die Angaben in der Datenbank vollständig sind, das Dokument in einem Verzeichnis liegt und die stabile URL zugewiesen wurde, ist der bereits im Beleg konzipierte Export an die Systeme OPUS und bibliographisch relevante Verbundsysteme (z. Bsp. SWB oder Datenbanken der Deutschen Bibliothek) zu testen. Soll der Server einmal in den Produktionsbetrieb gehen, muss auch seine Sicherheit gewährleistet sein. Hier wird von der Diplomarbeit keine perfekte Lösung erwartet, weil das zu viel Zeit kosten und vom eigentlichen Thema ablenken würde. Was sich aber bei der Realisierung durch einfaches Einschalten aktivieren lässt, sollte auch genutzt werden. Bestimmte Zugriffe sind eben nur dem Autor oder der SLUB gestattet; das sollte der Server weitgehend sicherstellen. Als dritter Baustein des Systems (nach Autoren- und SLUB-Eingabe) und als wichtigste Anwendung der Datenbank ist dann die Oberfläche zum Navigieren und

Suchen auf dem Server zu entwerfen und prototypisch zu realisieren. Die Navigation sollte eine hierarchische Struktur verwenden, die Fakultäten und Institute listet. Die Suche stützt sich auf Autorennamen, Titel, Schlagworte und ggf. weitere Metadaten. Es sind entsprechende Web-Seiten zu gestalten. Im Unterschied zum Großen Beleg erfolgen die Arbeiten jetzt direkt auf den Zielrechner in der SLUB. Dort ist das im Beleg erstellte System bereits installiert. Das gewählte Systemkonzept ist laufend zu überprüfen und an den Anforderungen zu messen. Alternative Konzepte sind der Literatur zu entnehmen und vergleichend zu bewerten. Die eingesetzten Werkzeuge, mit denen ja oft noch nicht sehr viel Erfahrung gemacht wurde, sind ebenfalls zu prüfen und zu bewerten.

Arbeitsschritte:

- Auswahl eines Datenbankverwaltungssystems
- Schemaentwurf
- Anpassung der Programme aus dem Beleg an die Nutzung der Datenbank, vor allem durch Anleitung der in der Abt. DV der SLUB für Programmierarbeiten einsetzbaren Auszubildenden.
- Entwurf einer Verzeichnisstruktur für die Dokumente
- Generierung der Frontdoors aus der Datenbank
- Entwurf und Realisierung von hierarchischen Strukturen für die Navigation
- Entwurf und Realisierung einer Suchmaske sowie der Darstellung von Ergebnislisten
- Dokumentation

2.2. Interpretation der Aufgabenstellung

Bei dieser Diplomarbeit handelt es sich um die inhaltliche Fortsetzung und Erweiterung einer Belegarbeit[Schb00], deshalb wird an vielen Stellen auch auf Erkenntnisse und Ergebnisse dieser Arbeit zurückgegriffen werden (mit dem Rückblick auf die Belegarbeit beschäftigt sich vor allem Abschnitt 2.3 auf Seite 33). Wie in Abschnitt 1.1 bereits kurz dargelegt, soll diese Fortsetzung dazu führen, dass alle für die erste Ausbaustufe des HSSS vereinbarten Dienste möglichst vollständig, aber eben prototypisch realisiert werden. Dies erst erlaubt das Testen und Verbessern der mit dem HSSS verbundenen Arbeitsabläufe. Wie aus Abschnitt 2.1 deutlich wird, gehören zu den Diensten des HSSS insbesondere drei unterscheidbare Teilanwendungen (siehe auch **Abbildung 1**):

- Die erste Teilanwendung beinhaltet die Unterstützung der Autoren bei der Übermittlung der Metadaten zu ihrer Hochschulschrift an die SLUB. Diese Teilanwendung wurde im Rahmen der Belegarbeit bereits vollständig realisiert.

- Die zweite Teilanwendung realisiert die administrative Verwaltung, Bearbeitung und Validierung aller Metadaten vom Eintreffen[1] bis hin zur persistenten Speicherung. Diese Teilanwendung wurde von der Belegarbeit nur zum Teil implementiert.

- Die dritte Teilanwendung soll den Zugriff auf alle Hochschulschriften und deren Metadaten im HSSS realisieren. Sie wurde in der Belegarbeit noch gar nicht behandelt und stellt einen thematischen Schwerpunkt in der Diplomarbeit dar.

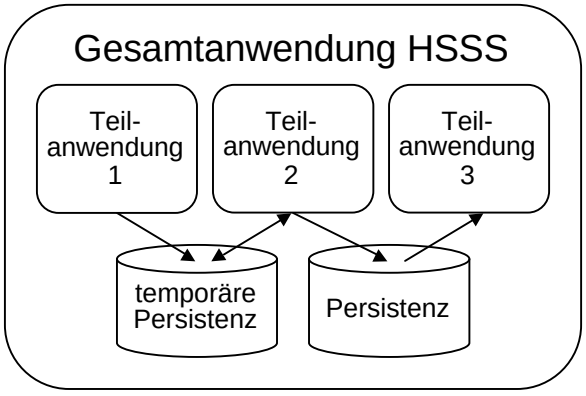

Abbildung 1 *- schematische Darstellung der HSSS-Gesamtanwendung*

[1] direkt nachdem sie vom Autoren mit Hilfe der ersten Teilanwendung erstellt wurden

Soweit es der zeitliche Rahmen der Diplomarbeit zulässt, sollen die schon in der Belegarbeit angesprochenen Verbesserungsmöglichkeiten berücksichtigt werden, um ein insgesamt möglichst stabiles und stimmiges System zu realisieren (siehe dazu auch Abschnitt 2.3). Einige Bereiche der Gesamtanwendung wie etwa die dritte Teilanwendung sind aber auch völlig neu ohne Rückgriff auf die Belegarbeit zu entwickeln, dazu gehört z. Bsp. die Umstellung der Persistenzrealisierung von einer dateibasierten Lösung (Beleg) auf eine datenbankbasierte Lösung (Diplom). Noch interessanter wäre eine Realisierung, die es ermöglicht, wahlweise dateibasiert oder datenbankbasiert zu speichern. Der in der Aufgabenstellung angesprochene Einsatz von XML-Technologie ist für alle Teilbereiche der Anwendung zu prüfen, insbesondere für solche mit Datenaustausch (z. Bsp. Metadatenexport an die entsprechenden Institutionen). Auch zur Präsentation von Zwischenergebnissen (z. Bsp. Teile der Metadaten) könnte sich diese Technologie eignen.

Ein wichtiges Ziel der Diplomarbeit sollte es sein, einen umfassenden Arbeitsablauf für den Gegenstandsbereich zu definieren und ihn detailliert auf seine Praxistauglichkeit hin zu überprüfen. Deshalb sollten, nachdem die erste und die zweite Teilanwendung fertig gestellt sind, alle damit verbundenen Arbeitsabläufe getestet werden[2]. Dazu gehört auch, dass die Lesbarkeit der exportierten Metadaten nachgewiesen werden kann. Um die Arbeitsabläufe zu festigen, sollten eine Reihe von schon vorhandenen „echten“ Daten in dieser Weise mit den zwei Teilanwendungen bearbeitet werden. Danach müssen die jetzt vollständigen Metadaten persistent in die Datenbank eingestellt werden. Zu diesem Zweck ist ein Datenbank-Schema zu entwickeln, welches insbesondere auch den Anforderungen der dritten Teilanwendung genügt. Außerdem ist die sogenannte „Frontdoor“ automatisch zu erstellen sowie das Dokument entsprechend einzustellen. Besondere Aufmerksamkeit gilt der geforderten Abbildung einer absoluten, sich nie mehr ändernden URL auf eine reale, im Nachhinein noch veränderbare URL. Diese Abbildung ist auf verschiedenen Wegen realisierbar, auf jeden Fall aber ein wichtiger Teil der Anwendung, da jeder Zugriff auf eine Ressource des HSSS auch eine solche Abbildung verlangt[3].

Als dritte Teilanwendung ist schließlich eine Applikation zu realisieren, welche den Zugriff auf die persistenten Daten ermöglicht. Dieser Zugriff könnte durch strukturierte Listen, Suchmöglichkeiten wie etwa die Feldsuche oder Volltextsuche beziehungsweise durch Navigationsverfahren realisiert werden. Treffer sollten immer in Form der oben angesprochenen absoluten URL präsentiert werden, erst der direkte Zugriff erfolgt über eine Indirektion auf die reale URL. Volltextsuche und Feldsuche könn-

[2] dies gilt insbesondere für Arbeitsabläufe, bei denen externe Institutionen involviert sind

[3] Möglicherweise handelt es sich bei diesem Teil der Anwendung um einen Flaschenhals der Gesamtanwendung

ten durch eine in die Datenbank integrierte Volltext-Suchmaschine realisiert werden, sofern diese vorhanden ist. Andernfalls müssen die Suchvorgänge durch die Anwendung direkt auf der Datenbank vorgenommen werden. In diesem Fall ist schon beim Design der Datenbank-Schemata darauf zu achten, entsprechende Optimierungen vorzunehmen. Eine dritte Möglichkeit besteht in der Integration einer externen Volltext-Suchmaschine. Die Navigation lässt sich wahrscheinlich am besten mit direktem Zugriff auf die Datenbank realisieren. Falls zeitlich möglich, wäre eine Messung und Auswertung des Antwortzeitverhaltens der dritten Teilanwendung bei großen Datenmengen bzw. vielen Zugriffen interessant, ebenso die Qualität der Suche.

2.3. Rückblick auf den Großen Beleg

Im Rahmen einer Belegarbeit[Schb00] wurden bereits wichtige Grundlagen erarbeitet und Ergebnisse und Erkenntnisse erzielt, auf die sich die Diplomarbeit vielfach beziehen kann und wird. Dieser Abschnitt soll sich mit den Anforderungen und den wichtigsten Ergebnissen der Belegarbeit beschäftigen, insbesondere im Hinblick auf deren Relevanz und Nutzbarkeit für die Diplomarbeit. Außerdem sollen die Ergebnisse kritisch hinterfragt und auf im Diplom realisierbare Verbesserungsmöglichkeiten hin untersucht werden.

2.3.1. Anforderungen und Ergebnisse der Belegarbeit

Zu den wichtigsten Anforderungen des Großen Beleges gehörte es, ein Verfahren zu entwickeln, das es Autoren von Hochschulschriften an der TU Dresden ermöglicht, diese Dokumente an die SLUB zu übergeben, um sie langfristig und gut erschlossen weltweit verfügbar zu machen. Hierfür galt es vor allem die technischen Voraussetzungen zu schaffen, indem am Anfang die möglichen technischen bzw. konzeptionellen Alternativen einer Realisierung verglichen wurden. Außerdem war eine Anwendung prototypisch zu entwickeln, die es den Autoren von Hochschulschriften erlaubt, die Metadaten zu ihren Dokumenten möglichst bequem an die SLUB zu übermitteln.

Zu den wichtigsten Ergebnissen des Beleges zählen deshalb die grundsätzlichen Technologieentscheidungen, die ausführlich begründet wurden und in der Regel auch für die Diplomarbeit verbindlich sind. Zu diesen Technologieentscheidungen gehört die Wahl der Entwicklungsplattform (Java) sowie der auf dieser Entwicklungsplattform basierenden API's für die Implementation von webbasierten Anwendungen (Servlet-API, JSP) und der Komponenten für deren Realisierung (*Apache* Webserver, *Tomcat* Servlet-Container)[Schb00]. Alle genannten Komponenten werden nach wie vor auch im Rahmen des Diploms in dieser Form Verwendung finden. Aber auch Antworten zu inhaltlichen Fragen wie Organisationsabläufe, Leistungsumfang eines Hochschulschriftenservers sowie Integrationsanforderungen und -möglichkeiten zählen zu den Ergebnissen des Beleges. Zusätzlich zu diesen grundlegenden Erkenntnissen liefert die Belegarbeit jedoch noch weitere Ergebnisse in Form von prototypisch implementierten Teilanwendungen (die inhaltlich den anfangs eingeführten Teilanwendungen 1 und 2 entsprechen). Diese Teilanwendungen konnten erfolgreich realisiert und getestet werden und bestätigten die getroffenen Technologieentscheidungen.

Durch die konsequente Verwendung von moderner, plattformunabhängiger Internettechnologie konnte schon im Rahmen der Belegarbeit eine Lösung aufgezeigt werden, die sich von etablierten Alternativen unterscheidet und vor allem in den Bereichen Skalierbarkeit, Erweiterbarkeit und Zukunftsfähigkeit teilweise deutliche Vorteile bietet. Ein großer Teil dieser Lösung kann auch ohne jegliche Abstriche für die Diplomarbeit verwendet und in die zu implementierenden Anwendungen integriert werden, ein weiterer Teil kann zumindest als Ausgangsbasis für weitere Entwicklungen dienen. Trotzdem gibt es natürlich auch einige Punkte, die verbesserungswürdig erscheinen, teilweise wurde auf diese Schwächen auch schon am Ende der Belegarbeit verwiesen. Sie sollen nach Möglichkeit im Rahmen des Diploms behoben werden. Der folgende Abschnitt geht noch einmal kurz auf einige Verbesserungsmöglichkeiten ein.

2.3.2. Verbesserungsmöglichkeiten

Zu den in der Belegarbeit angesprochenen Verbesserungsmöglichkeiten zählten eine Reihe von Diensten, die für eine Produktivversion eines Hochschulschriftenservers wünschenswert wären, aufgrund der geringen Zeitdauer und der akademischen Ausrichtung der Belegarbeit aber nicht realisiert werden konnten. Diese Punkte sind auch für das Diplom nicht von vordergründigem Interesse, da es nicht um die billige Realisierung eines Produktivsystems für die SLUB geht, sondern um die prototypische Implementierung technisch und akademisch interessanter Fragen, die auch übergreifend von Interesse sind. Erstgenannte Verbesserungen könnten aber durchaus von den in der Aufgabenstellung erwähnten Auszubildenden der SLUB mit entsprechenden Hilfestellungen realisiert werden.

Zu den im Diplom zu beachtenden, sich aus der Belegarbeit ergebenden Verbesserungsmöglichkeiten gehören vor allem die konsequente serverseitige Validierung der Metadaten in der ersten Teilanwendung[4] sowie die schon angesprochene datenbankbasierte Speicherung der Metadaten. Eine lohnende Weiterentwicklung wäre eine erhöhte Flexibilität der Anwendungen durch weitgehende Konfigurierbarkeit. In diesem Zusammenhang ist XML als mögliches Format von Konfigurationsstrukturen zu prüfen. Ebenso ist eine Integration von Kommunikationsstandards wie etwa Mail oder FTP in die Anwendungen zu überdenken. Die im Beleg erstellten Arbeitsabläufe sind noch detaillierter zu spezifizieren und in Zusammenarbeit mit den Institutionen SLUB, DDB (Die Deutsche Bibliothek) und SWB (Südwestdeutscher Bibliotheksverbund) in die Praxis umzusetzen. Bei der Überarbeitung der vorhandenen Implementationen

[4] hier wurde in der Beleg-Version zum Teil nur Client-seitig mit Hilfe von JavaScript validiert

und vor allem bei dem Design neuer Teilanwendungen sollen die vorhandenen objektorientierten Strukturen noch verbessert werden, insbesondere ist die Verwendung und Integration von Entwurfsmustern zu überdenken. Weitere Verbesserungsmöglichkeiten ergeben sich unter Umständen aus der Betrachtung bereits existierender Hochschulschriftenserver bzw. durch die Auswertung der Arbeit von Organisationen und Initiativen im Umfeld digitaler Publikationen / Archive, insbesondere im Hinblick auf existierende Standards und zu beachtende Empfehlungen. Mit dieser Thematik wird sich der folgende Abschnitt auseinandersetzen.

2.4. Digitale Archive und elektronisches Publizieren im bibliothekarischen Umfeld

Wie in Kapitel 1 im Abschnitt zur Motivation bereits dargelegt wurde, müssen sich vor allem die Universitätsbibliotheken in Zukunft neuen Herausforderungen stellen. Durch den Siegeszug von Computer und Internet hat sich die Art und Weise des Publizierens (insbesondere an wissenschaftlichen Einrichtungen) stark verändert und wird sich weiter stark verändern.

Um sich diesen Herausforderungen stellen zu können, müssen sich die Bibliotheken verstärkt mit den neuen technischen Möglichkeiten auseinandersetzen und in diesen Bereichen „Know-How“ aufbauen[Wis01]. Der Aufbau eines Dokumentenservers für Hochschulschriften und das Angebot der damit verbundenen Dienste stellt einen guten Einstieg in diese Materie dar, der von einigen Institutionen auch bereits gewagt wurde. In Deutschland gab es eine Reihe von Projekten[5], die sich mit dieser Problematik auseinandersetzten und die Grundlagen schufen, auf welchen heute jede Bibliothek aufbauen kann. Auch der Großteil an technischen Grundsatzfragen ist heutzutage bereits geklärt, so dass es jetzt vor allem auf den Willen der Bibliotheken ankommt, in ihre Zukunft zu „investieren“, und natürlich auch darauf, dass die Bibliotheken bei dieser Herausforderung entsprechend unterstützt werden[6].

Allerdings gibt es auch eine Reihe technischer Detailfragen, die zwar seit Jahren bekannt sind und rege diskutiert werden, die jedoch noch längst nicht zufriedenstellend gelöst sind. Dazu gehört das große Problemfeld der physischen Langzeitsicherung digitaler Daten[Che01]. Vor allem gibt die geringe Haltbarkeit bisheriger und derzeitiger Datenträger zu denken, zum einen, was die Haltbarkeit der auf ihnen gespeicherten Daten angeht, zum anderen aber auch ihre technologische Relevanz und Verbreitung

[5] unter anderem:
- „Dissertationen Online“, DFG-Projekt 1998-2000
- „Networked European Deposit Library - NEDLIB“, Europäische Kommission 1998-2000

[6] Diese Aufgabe kommt vor allem Bund und Ländern zu. Von fundamentaler Bedeutung ist dabei die Grundsatzfrage, ob wissenschaftliche Veröffentlichungen eine Ware bzw. Dienstleistung sind, oder aber ein öffentliches Gut. Leider ist diese Grundsatzfrage bis heute (auch politisch) nicht endgültig geklärt. Allerdings gibt es einige kaum zu übersehende Argumente für wissenschaftliche Publikationen als frei zugängliches, öffentliches Gut (was keine Beeinträchtigung etwaiger Urheberrechte einschließt) und gegen das Modell eines beschränkten, kostenpflichtigen Zugangs, der dann nur zahlenden Wissenseliten verfügbar wäre und wo die zu großen Teilen vom Steuerzahler finanzierte Wissenschaft sich ihre eigenen 'Produkte' von der Wirtschaft zurückkaufen müsste.[Sie00b]

an sich[7][Gro00]. Vergegenwärtigt man sich, dass momentan gebräuchliche Datenträger auch im geschichtlichen Vergleich ganz weit hinten liegen (was deren Haltbarkeit betrifft), wird deutlich, in welchem Dilemma sich digitale Archive weltweit befinden (siehe auch **Tabelle 1**).

Medium	Haltbarkeit in Jahren
Ägyptische Steintafeln	min. 2200
Säurefreies Papier	100 bis 500
Microfilm	10 bis 500
CD-ROM	5 bis 200
Magnetband	10 bis 30
Digitalband	10 bis 30
VHS-Band	10 bis 30
Zeitungspapier	10 bis 20

Tabelle 1 - Lebensdauer verschiedener Datenträger[8]

Dabei ist das Problem gar nicht so sehr technischer Natur - da nach Ansicht vieler Experten das Know-How, langlebige Datenträger zu entwickeln, längst vorhanden ist - sondern vielmehr ein Problem des fehlenden Bewusstseins: kaum jemand denkt in Bezug auf digitale Archivierung in den richtigen (sehr großen) Zeitdimensionen. So bleibt für die Praxis nur eine Strategie, die auch die alten Ägypter vor Jahrtausenden schon anwandten: das Kopieren. Aber selbst wenn diese Probleme praktisch gelöst und langlebige Datenträger für die digitale Archivierung verfügbar wären, bliebe ein weitaus schwierigeres bestehen: das der unterschiedlichen Formate. Schon heute sind eine Vielzahl von Datenformaten nach geschichtlich gesehen erschreckend kurzer Zeit (ca. 5 - 40 Jahren) bereits nicht mehr lesbar, die darin kodierten Informationen also verloren.

Auch für die Zukunft ist abzusehen, dass nur einige wenige der heute gebräuchlichen Datenformate mittelfristig lesbar bzw. verarbeitbar bleiben werden. Kurzfristig hilft hier nur, für die Archivierung möglichst weitverbreitete, einfache Formate zu verwenden und auf Komprimierung etc. zu verzichten. Langfristig kann nur ein für Konverter publiziertes Formatschema bzw. ein einheitliches Datenformat bzw. Dateiformat für die digitale Archivierung Abhilfe schaffen[9][Gro00]. Auf diesem interessanten Gebiet sollten vor allem auch die Bibliotheken aktiv werden und bleiben, da hier Lösungen für die Zukunft gerade auch für sie dringend erforderlich sind.

[7]So sind etwa für die Anfang der 90-er Jahre als Archivierungsmedium auch in Bibliotheken noch gebräuchlichen 5 1/4 Zoll Disketten kaum noch Laufwerke verfügbar, die diese lesen können.

[8]Quelle: National Media Laboratory, Kodak, 1998

[9]Universal Preservation Format (UPF), http://info.wgbh.org/upf

Der Aufbau eines Dokumentenservers und eines digitalen Archives mit Blick auf die angesprochenen Schwierigkeiten, dessen Betrieb und die Einführung einer geeigneten Ablauforganisation in den einzelnen Universitätsbibliotheken sind aber nur der Anfang. Die dergestalt entstandenen dezentralen Systeme benötigen gemeinsame Schnittstellen und ein gemeinsames Protokoll, um über sich selbst hinaus übergreifend von Nutzen zu sein. Damit das (auch im Nachhinein) mit einem vertretbaren Aufwand möglich ist, sollte bei dem Aufbau der entsprechenden Systeme von Anfang an darauf geachtet werden, dass so weit wie nur irgend möglich etablierte Standards berücksichtigt und propritäre Lösungen vermieden werden. Für die Definition solcher Standards, die insbesondere den für Bibliotheken interessanten technischen Bereich abdecken, gibt es eine Reihe von Organisationen und Initiativen, die im Folgenden kurz vorgestellt werden sollen.

2.4.1. Open Archives Initiative (OAI)

Die *Open Archives Initiative* (OAI)[L-OAb] hat sich zum Ziel gesetzt, Standards für die Zusammenarbeit, Kommunikationsfähigkeit und den Datenaustausch zu entwickeln, die es erlauben, digitale Inhalte effizient auszutauschen und zu verbreiten. Seine Wurzeln hat die Initiative vor allem im wissenschaftlichen Bereich, wo sie den Zugriff auf „e-Print“-Archive verbessern und erweitern möchte, um somit die gesamte wissenschaftliche Kommunikation zu bereichern. Die Mechanismen allerdings, auf denen die von der OAI entwickelten Spezifikationen basieren, sind sowohl von der Umgebung[10] als auch vom Inhalt der auszutauschenden Daten unabhängig.

Die OAI ist eine in den USA anerkannte Organisation, der zwei namhafte Professoren der *Cornell University* vorstehen. Sie besteht aus einem organisatorischen und einem technischen Komitee. Unterstützt wird die OAI von der *Digital Library Federation* und der *Coalition for Networked Information* sowie einer Reihe weiterer Organisationen und Universitäten in den USA. Hauptentwicklung der OAI ist das *Open Archives Metadata Harvesting Protocol*, welches ein generisches Interaktions-Framework für den verteilten Zugriff auf digitale Inhalte darstellt. Dieses Protokoll wurde mit Absicht sehr einfach gestaltet, um die Barriere einer Teilnahme bzw. einer Implementation möglichst gering zu halten. Dieses Framework kann jede Institution nutzen, um über das OAI-Protokoll an einem Peer-to-Peer-Verbund mit anderen Institutionen teilzunehmen, es müssen vorher lediglich die dafür notwendigen technischen Voraussetzungen geschaffen werden. Die Teilnahme kann zwei verschiedene

[10] insbesondere auch von der ökonomischen Umgebung

Dimensionen beinhalten:

1. Sogenannte „*Data Provider*“ unterstützen das OAI-Protokoll dergestalt, dass sie Metadaten-Informationen über ihre Daten und Inhalte (Archive) zur Verfügung stellen. Zusätzlich können sich solche Institutionen bei der OAI registrieren lassen, um den Fakt, dass sie in dieser Form das OAI-Protokoll unterstützen, öffentlich zu machen.

2. Sogenannte „*Service Provider*“ verteilen Dienstanfragen an die *Data Provider* und nutzen die erhaltenen Metadaten als Basis für Mehrwertdienste. Auch ein *Service Provider* kann sich bei der OAI registrieren, um seine Existenz zu publizieren.

Daraus wird schon deutlich, dass es für eine Bibliothek, die zum Beispiel einen eigenen Dokumentenserver für digitale Hochschulschriften aufbaut, durchaus Sinn macht, sich mit dem *Open Archives Metadata Harvesting Protocol* vertraut zu machen, um gegebenenfalls als *Data Provider* an einem dezentralen Verbund von verschiedenen Archiven teilnehmen zu können, indem man Metadaten über die eigenen Inhalte zur Verfügung stellt. Eine Bibliothek könnte natürlich auch als *Service Provider* auftreten, indem sie zum Beispiel eine Metasuche über ihr eigenes Archiv und das beteiligter Institutionen ermöglicht.

Die Idee, verschiedene lokale Archive über ein gemeinsames Protokoll miteinander zu verbinden und somit virtuell einen globalen Zugriff zu realisieren, ist natürlich alles andere als neu. So gibt es schon seit geraumer Zeit sogenannte *„Harvesting Protocols“* für digitale Dokumente der Fachrichtungen Physik und Mathematik[11]. Außerdem gibt es seit langer Zeit komplexe Protokolle wie etwa Z39.50, das insbesondere im Bibliothekswesen eingesetzt wird. Diese vorhandenen Möglichkeiten haben leider jeweils spezifische Schwächen, die eine ausreichende Verbreitung bisher verhinderten. So beziehen sich die oben angesprochenen *Harvesting*-Architekturen auf bestimmte wissenschaftliche Disziplinen, sind also eher „Graswurzel-Initiativen“ und nicht generisch genug, um übergreifend genutzt werden zu können. Das Protokoll Z39.50 hingegen ist viel zu komplex und ebenfalls zu spezifisch, um eine größere Verbreitung zu finden. Im folgenden Abschnitt sollen kurz einige technische Details des *Open Archives Metadata Harvesting Protocol* in der Version 1.0 näher erläutert werden, wobei speziell die Frage von Interesse ist, wie die angesprochenen Schwächen der vorhandenen Lösungsansätze vermieden werden sollen.

[11] - MathNet: http://www.math-net.de
- PhysNet: http://www.physik.uni-oldenburg.de/EPS/PhysNet/

2.4.1.1. Protocol for Metadata Harvesting

Das Hauptziel des *Open Archive Protocol for Metadata Harvesting* (im Folgenden als OAI-Protokoll bezeichnet)[L-OAc] ist die Realisierung eines anwendungsunabhängigen Interaktions-Frameworks, welches von den verschiedensten Organisationen einfach genutzt werden kann, die sich im weitesten Sinne mit dem Publizieren von Inhalten über das Web beschäftigen. Am Anfang sollen kurz einige Protokoll-Begriffe definiert werden:

- Ein **Repository** bezeichnet einen netzwerkfähigen Server, an den in das Protokoll HTTP eingebettete OAI-Protokoll-Anfragen gesendet werden können.
- Ein **Record** ist eine in XML kodierte Byte-Folge, die von einem *Repository* als Antwort auf eine OAI-Protokoll-Anfrage zurückgesendet wird und die einem Archivinhalt eindeutig zugeordnet ist.
- Ein **Unique Identifier** ist ein Schlüssel für den Zugriff auf die Metadaten eines durch diesen Schlüssel eindeutig bestimmten Archivinhalt des *Repository*.
- Ein **Datestamp** kann das Datum der Erstellung, des Entfernens oder der letzten Änderung eines Archivinhalts sein.
- Ein **Set** ist ein optionales Konstrukt, das der Gruppierung von Archivinhalten eines *Repository* zum Zwecke eines selektiven Zugriffs dient.

Ein *Repository* beinhaltet eine Menge von Dokumenten (digitale Archivinhalte) und deren Metadaten. Diese Metadaten sind über das OAI-Protokoll verfügbar. Eine Anfrage an das *Repository* zu einem Dokument wird durch einen *Record* beantwortet, welcher in XML kodierte Metadateninformationen über das Element enthält. Die *Record*-Struktur besteht aus den folgenden Teilen (siehe auch **Listing 1**):

- Dem *„Header“*-Abschnitt: Informationen, die für die Bearbeitung der Informationen notwendig sind, wie z. Bsp. ein *Unique Identifier* und *Datestamp*
- Dem *„Metadata“*-Abschnitt: Die Metadateninformationen für das Element in einem bestimmten Metadatenformat. Das OAI-Protokoll erlaubt die Beschreibung mit verschiedenen Metadatenformaten, ein *Repository* muss aber zumindest in der Lage sein, ein *Record* mit im *Dublin Core* Metadatenformat[L-DC] beschriebenen Metadaten zurückzuliefern
- Dem *„About“*-Abschnitt: Ein optionaler Abschnitt, der Informationen über den vorangegangenen *Metadata*-Abschnitt enthält, typischerweise rechtliche Bedingungen oder Nutzungshinweise

```
<header>
  <identifier>oai:hsss:99010018976−5678</identifier>
  <datestamp>2001−01−01</datestamp>
</header>
<metadata>
  <dc xmlns="http://purl.org/dc/elements/1.1/"
   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema−instance"
   xsi:schemaLocation="http://purl.org/dc/elements/1.1/
   http://www.openarchives.org/OAI/dc.xsd">
    <title>Quo Vadis Quanten?</title>
    <creator>Quark, T.</creator>
    <creator>Sprung, Q.</creator>
    <description>We simulate the center of mass motion of cold atoms in a
     standing, amplitude modulated, laser field as an example of a system
     that has a classical mixed phase−space.</description>
    <date>2001−01−01</date>
    <type>doctoral</type>
    <identifier>http://hsss.slub−dresden.de/hsss/servlet/hsss.urlmapping.MappingServlet?id=99010018976−5678
    </identifier>
  </dc>
</metadata>
<about>
  <ea xmlns="http://hsss.slub−dresden.de/eprints−about"
   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema−instance"
   xsi:schemaLocation="http://hsss.slub−dresden.de/eprints−about
   http://hsss.slub−dresden.de/eprints−about.xsd">
    <archive>SLUB HSSS Archive</archive>
    <usage>Metadata may be used without restrictions as long as the OAI
     identifier remains attached to it.</usage>
  </ea>
</about>
```

***Listing 1** : Record-Struktur des OAI-Protokolls*

Listing 1 zeigt verkürzt die Struktur eines fiktiven *Record* für den HSSS. Hier handelt es sich um eine Dissertation zu einem physikalischen Thema. Der *Header*-Abschnitt enthält den für die Bearbeitung durch das OAI-Protokoll notwendigen *Unique Identifier*, der zum Teil aus dem Metadaten-Identifier besteht, außerdem ein *Datestamp*. Der *Metadata*-Teil enthält Metadaten zu der fiktiven Dissertation, die lediglich im Metadatenformat DC (Dublin Core) beschrieben sind. Dies stellt die Minimallösung dar. Optional könnten die Metadaten zu diesem Dokument im *Metadata*-Abschnitt durch eine Reihe weiterer Metadatenformate beschrieben werden. Der optionale *About*-Abschnitt besteht aus der Bezeichnung des Archivs (bzw. des *Repository*) und aus Nutzungshinweisen.

OAI-Protokoll-Anfragen werden in HTTP-Anfragen eingebettet. Dadurch kann eine Implementation dieses Protokolls normale Standardsoftware benutzen, zum Beispiel einen Webserver wie *Apache*, der so konfiguriert ist, dass er OAI-Anfragen an eine Anwendung weiterleitet, die diese Anfragen verarbeiten kann[12]. Grundsätzlich beste-

[12]Zum Beispiel ein Perl-Script über CGI oder ein PHP-Script, besser eine plattformunabhängige Alternative wie etwa ein Servlet

hen OAI-Anfragen - genau wie HTTP-Anfragen auch - aus einer Ziel-URL und einer Reihe von Schlüssel-Wert-Paaren (Parametern). Zum Beispiel beschreibt die folgende OAI-Protokoll-Anfrage

```
http://hsss.slub-dresden.de/hsss/servlet/hsss.oai.OAIServlet?verb=GetRecord
_ &identifier=oai:hsss:99010018976-5678&metadataPrefix=oai_dc
```

eine *„GetRecord“*-Anfrage an die URL „hsss.slub-dresden.de/hsss/servlet/hsss.oai.-OAIServlet“ (Anwendung des *Repository*). Als Parameter werden in Form von Schlüssel-Wert-Paaren zum einen die Art der Anfrage (*GetRecord*) sowie der Identifier des gesuchten Elementes übergeben, zusätzlich das Metadaten-Prefix des Metadatenformates[13]. Die Einbettung von OAI-Anfragen ist sowohl in HTTP-GET-Anfragen (wie gezeigt) als auch in HTTP-POST-Anfragen möglich.

OAI-Protokoll-Antworten werden als HTTP-Antworten versendet und haben daher deren typische Merkmale und Struktur. Da OAI-Antworten XML-Daten enthalten (wie am Beispiel der *Record*-Struktur zu sehen war), muss der HTTP-Header *„Content-Type“* auf *text/xml* gesetzt werden (**Listing 2** zeigt den schematischen Aufbau einer OAI-Protokoll-Antwort).

```
1  HTTP/1.0 200 OK
2  Content−Type: text/xml
3  ...
4
5  <?xml version="1.0" encoding="UTF−8" ?>
6  <GetRecord
7   xmlns="http://www.openarchives.org/OAI/1.0/OAI_GetRecord"
8   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema−instance"
9   xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_GetRecord
10  http://www.openarchives.org/OAI/1.0/OAI_GetRecord.xsd">
11    <responseDate>2001−05−01T19:20:30−04:00</responseDate>
12    <requestURL>http://hsss.slub−dresden.de/hsss/servlet/hsss.oai.OAIServlet?verb=GetRecord
13      &amp;identifier=oai%3hsss%3A99010018976−5678
14      &amp;metadataPrefix=oai_dc</requestURL>
15    <..content of record..>
16  </GetRecord>
```

***Listing 2** : Struktur einer OAI-Protokoll-Antwort*

Eine Reihe von OAI-Anfragen liefern Listen von diskreten Elementen. Unter Umständen soll nicht die gesamte Liste auf einmal übertragen werden (zum Beispiel ab einer bestimmten Listengröße). In diesem Fall bedarf es einer Flusskontrolle durch das Protokoll. Im OAI-Protokoll kann ein *Repository* auf eine Anfrage mit einer unvollständigen (Teil-)Liste antworten, allerdings nur zusammen mit einem Wiederaufnahmezeiger (*Resumption Token*). Um die vollständige Liste zu erhalten, muss der Client

[13]enthaltene Sonderzeichen wie „:“ müssen selbstverständlich kodiert werden

(gegebenenfalls mehrfach) dieselbe Anfrage inklusive des Wiederaufnahmezeigers an das *Repository* schicken. Im folgenden sollen die wichtigsten OAI-Protokoll-Anfragen[14] nebst den korrespondierenden Protokoll-Antworten vorgestellt werden.

- **GetRecord**
  Diese Anfrage wird genutzt, um einen einzelnen *Record* eines Elementes des *Repository* zu erhalten. Obligatorische Parameter sind der Schlüssel (**identifier**) des gewünschten *Record* sowie das Metadatenformat (**metadataPrefix**), welches die Antwort enthalten soll.

- **Identify**
  Diese Anfrage wird genutzt, um Informationen über ein *Repository* zu erhalten, zusammen mit Informationen über die Administration und die betreibende Institution. Diese Anfrage benötigt keine Parameter.

- **ListIdentifiers**
  Diese Anfrage wird genutzt, um die Schlüssel (identifier) der *Records* zu erhalten, die ein *Repository* enthält. Optionale Parameter können die Menge der Schlüssel einschränken, z. Bsp. auf Basis der Zugehörigkeit zu einem bestimmten *Set* oder aufgrund der Modifikation oder Erstellung innerhalb eines bestimmten Zeitraumes.

- **ListMetadataFormats**
  Diese Anfrage wird genutzt, um die verfügbaren Metadatenformate eines *Repository* zu erhalten. Der optionale Parameter **identifier** verringert die Ergebnismenge auf das Metadatenformat für einen bestimmten *Record*.

- **ListRecords**
  Diese Anfrage wird genutzt, um *Records* eines *Repository* zu erhalten. Der obligatorische Parameter **metadataPrefix** bezeichnet das gewünschte Metadatenformat, optionale Parameter können die Ergebnismenge nach verschiedenen Kriterien einschränken.

- **ListSets**
  Diese Anfrage wird genutzt, um eine *Set*-Struktur eines *Repository* zu erhalten.

Zusammenfassend lässt sich sagen, dass das OAI-Protokoll in seiner jetzigen Form ein schlankes, leicht zu implementierendes Protokoll ist, welches sich aufgrund dieser

[14] Wie bereits gezeigt, wird jede OAI-Anfrage über das HTTP-Protokoll verschickt, indem bestimmte Schlüssel-Wert-Paare übergeben werden, z. Bsp. der Parameter *„verb“* für die Bezeichnung der OAI-Anfrage

Tatsachen und der Nutzung von HTTP als Trägerprotokoll weiter verbreiten sollte als die vorher angesprochenen Alternativen. Ein Vorteil dieses Protokolls ist sicherlich auch, dass die Anforderungen an beteiligte Archive relativ gering sind; eigentlich ist lediglich die Beschreibung der Inhalte durch das Dublin Core Metadatenformat zwingend vorgeschrieben. Für die Zukunft wäre es wünschenswert, das Protokoll komplett XML-basiert zu gestalten[15] und eine moderne Referenz-Implementation (z. Bsp. in Form eines Servlets) anzubieten.

2.4.2. Deutsche Initiative für Netzwerkinformationen (DINI)

Die Deutsche Initiative für Netzwerkinformationen (im Folgenden nur noch als DINI bezeichnet)[L-DIa] ist eine Initiative von vier Partnerorganisationen (u.a. Deutscher Bibliotheksverband Sektion 4: Wissenschaftliche Universalbibliotheken) und sieht es als ihre zentrale Aufgabe an, angesichts der zunehmend flächendeckenden und in ihrer Leistungsfähigkeit stark ansteigenden Vernetzung sowie dem zunehmenden Übergang von der traditionellen papiergebundenen Publikation und Kommunikation zu einem modernen, elektronisch basierten und damit effektiven Informationsmanagement den Wandel der Informationsinfrastruktur innerhalb und zwischen den deutschen akademischen Bildungseinrichtungen voranzutreiben. Dabei steht für die DINI der Aufbau von allgemein zugreifbaren, effektiv recherchierbaren, systematisch gepflegten und authentisierten lokalen Hochschularchiven im Vordergrund.

Aus diesem Grund setzt sich die DINI sehr für die Einhaltung, aber auch für die gemeinsame Entwicklung und Etablierung von Standards ein. Am 12. September 2000 hat die DINI in einem „DINI Appell“ alle Betreiber von lokalen elektronischen Archiven an deutschen Hochschulen oder Förderinstitutionen aufgerufen, die Spezifikation der OAI zu implementieren. Als Begründung verweist die DINI in ihrem Appell darauf, dass ein Großteil der in den letzten Jahren auf Fachbereichs-, Hochschul- oder Landesebene entstandenen elektronischen Archive nicht nach einem abgestimmten, kompatiblen und auf hohe Verfügbarkeit orientierten Konzept aufgebaut wurde; so dass eine gemeinsame Nutzung mehrerer Archive gar nicht oder nur eingeschränkt (und mit erheblichem Aufwand verbunden) möglich sei. In vielen Fällen ist sowohl die Persistenz der Datensätze als auch deren Beschreibung in Metadatensätzen kaum gegeben. Da dieser Zustand die nationale und internationale wissenschaftliche Kommunikation in erheblichem Maße behindert, bedarf die Situation einer möglichst raschen Verbesserung, die nach Ansicht der DINI am ehesten durch die Implementa-

[15] hier bietet sich insbesondere SOAP (Simple Object Access Protocol) an[L-SO]

tion des OAI-Standards gegeben ist. Daher empfiehlt die DINI unter anderem auch den Einrichtungen, die in Deutschland über die Förderung von Projekten im Zusammenhang mit wissenschaftlicher Kommunikation entscheiden, die Einhaltung der OAI-Spezifikationen zur Voraussetzung für die Bewilligung solcher Förderanträge zu machen.

Innerhalb der DINI gibt es verschiedene Arbeitsgruppen, die sich mit jeweils unterschiedlichen Schwerpunktthemen beschäftigen. Von besonderem Interesse für den Bereich von wissenschaftlichen Dokumentenservern und Archiven im allgemeinen und damit auch für den HSSS im besonderen ist dabei die DINI-Arbeitsgruppe *„Informationsdienste und Standards für das elektronische Publizieren an Hochschulen“* (im Folgenden nur noch als Arbeitsgruppe *„Elektronisches Publizieren“* bezeichnet)[L-DIb].

2.4.2.1. Ziele der Arbeitsgruppe „Elektronisches Publizieren“

Ziel der Arbeitsgruppe *„Elektronisches Publizieren“* und des von ihr herausgegebenen Dokuments *„Empfehlungen zum Umgang mit elektronischen Hochschulpublikationen innerhalb von Universitäten, Bibliotheken und Verbünden“* (im Folgenden als Empfehlungen bezeichnet) [DIN01] sind Empfehlungen zum Umgang mit elektronischen Hochschulschriften. Dabei sollen zu diesem Thema sowohl der gegenwärtige Stand in Deutschland, aber auch die internationalen Entwicklungen reflektiert werden[16], wobei es der Arbeitsgruppe darum geht, eine „Kultur des elektronischen Publizierens“ an wissenschaftlichen Einrichtungen zu etablieren. Der folgende Abschnitt soll sich vor allem mit den in diesen Empfehlungen spezifizierten technischen und organisatorischen Anforderungen und Empfehlungen an Hochschulschriftenserver kritisch auseinandersetzen.

[16]Zwar handelt es sich bei den Empfehlungen erst um eine vorläufige Version und der Fokus liegt auf elektronischen Hochschulschriften und deren Behandlung, trotzdem kommen einige übergeordnete internationale Entwicklungen wie etwa das oben eingeführte OAI-Protokoll leider gar nicht vor oder deutlich zu kurz.

2.4.2.2. Anforderungen an einen Hochschulschriftenserver

Die Empfehlungen beginnen mit der Definition einiger Begriffe im Umfeld und versucht im Anschluss eine Motivation, welche die neue Arbeitsweise im Umgang mit digitalen Dokumenten mit der traditionellen, papiergebundenen Arbeitsweise vergleicht. Als wesentliche Vorteile werden der schnelle Zugriff auf die digitalen Dokumente sowie deren weltweite Verbreitung angesehen. Außerdem seien genauere Retrievalergebnisse aufgrund der besseren inhaltlichen Erschließung dieser Dokumente zu erwarten. Die Empfehlungen mahnen außerdem ein anderes Vorgehen bei der Erstellung wissenschaftlicher Dokumente an; nicht mehr das Präsentationsformat (Ausdruck), sondern die semantische Struktur der Dokumente sollte im Vordergrund stehen, da erst dieses Vorgehen eine flexible Mehrfachverwendung der elektronischen Publikation im Internet-, Druck- und Datenbankbereich ermöglicht. Solange für die Erstellung allerdings gängige Software wie etwa MS Word verwendet wird, dürfte die angesprochene Umstellung ausbleiben.

Zu den allgemeinen Anforderungen an das elektronische Publizieren zählen die Empfehlungen eine Reihe von Prinzipien in den Bereichen Formate, Standards, Protokolle, Authentizität, Metadaten, Retrieval und Langzeitarchivierung. Im ersten Teilabschnitt, der sich Formaten, Standards und Protokollen widmen soll, ist lediglich etwas von der Unbrauchbarkeit der meisten gebräuchlichen Textverarbeitungssysteme in Hinblick auf flexible Publikationen zu lesen, außerdem wird kurz auf die besonderen Schwierigkeiten in Bezug auf Multimediadokumente eingegangen. Hier wird als besonders kritisch eingeschätzt, dass sowohl die Zahl der sich am Markt befindlichen Autorensysteme als auch die Zahl der Multimediaformate inzwischen fast schon unüberschaubar geworden ist. Daraus resultieren sowohl Probleme für den Autor eines Multimediadokuments (die Wahl eines adäquaten Autorensystems) als auch für den Betreiber des entsprechenden Dokumentenservers (Formatunterstützung, Persistenzrealisierungen etc.). Der Abschnitt schließt mit dem Hinweis, möglichst offene Standards und speziell bei der Archivierung strukturierte Formate wie SGML bzw. XML zu verwenden. Im Bereich Authentizität geht es der Arbeitsgruppe vor allem darum, die Urheberschaft der wissenschaftlichen Publikation vor Fälschungen und Manipulationen gesichert zu sehen. Um dies gewährleisten zu können, sollen Autor, Inhalt und Veröffentlichungszeitpunkt der Publikation vom Dokumentenserver signiert werden. Das setzt natürlich die Einbindung in eine Public Key Infrastructure (PKI) und die Nutzung digitaler Zertifikate voraus. Ein zweiter Punkt ist die Sicherung des gesamten Dokumentenservers vor unbefugtem Zugriff und Manipulationen.

Im Folgenden gehen die Empfehlungen auf die Bedeutung von Metadaten für die langfristige Archivierung von digitalen Dokumenten ein. Hier taucht zum ersten und einzigen Mal ein Hinweis auf die *Open Archives Initiative* auf. So heißt es wörtlich: *„Die Einbindung in vernetzte Strukturen und Retrievalsysteme setzt voraus, dass [...] eine deutschlandweite Einigung zu elektronischen Hochschulpublikationen existiert oder die Empfehlungen der OAI genutzt werden“*[17]. Nach einer kurzen Definition von Metadaten wird vor allem der Dublin Core Element Set näher erläutert. Offen bleibt für den Leser, ob es womöglich ein deutschlandweit gültiges Schema für die Definition von Metadatensätzen nach Dublin Core gibt. Der sich anschließende Retrieval-Abschnitt gibt einige typische Recherchestrategien vor, die Dokumentenserver unterstützen sollten. Hierbei handelt es sich neben Navigation bzw. Browsing im Wesentlichen um die Freitextsuche, bei der in den Metadaten der Dokumente gesucht wird, um die Feldsuche, die eine erweiterte Form der Freitextsuche darstellt, sowie um die Volltextsuche, die sich auf den Volltext des Dokuments bezieht. Auch die Metasuche wird kurz erwähnt, allerdings nur im Zusammenhang mit dem Z39.50-Protokoll und leider nicht in Hinblick auf die OAI-Alternative. Ähnlich den Erörterungen am Anfang des Kapitels zur Problematik erfolgreicher Archivierung kommt auch die Arbeitsgruppe zu dem Schluss, dass im Sinne einer möglichst erfolgreichen und stabilen Langzeitarchivierung die zu wählenden Multimediaformate möglichst flexibel und strukturiert sowie im besten Fall auch hard- und softwareunabhängig sein sollten. Als mittelfristiger Kompromiss gilt auch hier die Nutzung von möglichst weit verbreiteten Formaten.

Es folgt ein mit *„Empfehlungen an die Hochschulen“* überschriebener Abschnitt, in dem es im Wesentlichen um organisatorische und rechtliche Belange geht. Ein interessanter Teil dieses Abschnitts beschäftigt sich mit der Notwendigkeit von Weiterbildungsmaßnahmen, und zwar sowohl für die Autoren als auch für das Dienstleistungspersonal von Bibliotheken oder Rechenzentren. Da es für eine wirklich effektive Archivierung und flexible Präsentation notwendig ist, zunehmend XML- respektive SGML-basiert zu publizieren, müssen die Autoren speziell in diesem Bereich geschult und beraten werden. So könnten Formatvorlagen für gängige Textverarbeitungssysteme entwickelt und eingeführt werden, die eine spätere Konvertierung in ein strukturiertes Format erleichtern, aber auch die direkte Erstellung von XML- bzw. SGML-Dokumenten sollte durch entsprechende Schulungen gefördert werden. Außerdem sollte auch der an der jeweiligen Institution etablierte Workflow im Zusammenhang mit dem digitalen Publizieren auf Informationsveranstaltungen regelmäßig vorgestellt und erläutert werden. Für das Dienstleistungspersonal dürfte der Weiterbildungsbedarf in den meisten Fällen noch größer sein. So wird in dem Empfehlungspapier der

[17] Dies schreibt die Arbeitsgruppe einer Initiative, die bereits über ein halbes Jahr vorher in einem eindringlichen Appell die Verwendung der OAI-Empfehlungen und des OAI-Protokolls angemahnt hat.

DINI-Arbeitsgruppe folgerichtig auch fast jeder der vorgestellten Technologiebereiche in die Rubrik „Weiterbildung“ eingeordnet, besonders die Bereiche Konvertierung, Umgang mit digitalen Signaturen, Zertifikaten und PKI sowie Metdatendatenanwendungen (hier speziell *Dublin Core*).

Den umfangreichsten Teil des Dokuments stellt der siebente Abschnitt dar, der mit *„Technologie - konkrete Beispielgeschäftsgänge“* überschrieben ist. Hier sollen insbesondere auch die schon in vorangegangenen Abschnitten beschriebenen Vorgehensweisen konkretisiert und praktisch veranschaulicht werden. Leider wird in vielen Fällen das vorangegangene lediglich wiederholt bzw. noch weiter vertieft, was in einem gewissen Widerspruch zum Titel des Abschnitts steht. Interessant sind die vorgestellten Beispielgeschäftsgänge einiger Institutionen, die sehr gut als Ausgangspunkt für die Entwicklung eigener Geschäftsgänge bzw. Arbeitsabläufe verwendet werden können. Danach werden Anforderungen an die Formate eines Archivs gestellt. Zu diesen Anforderungen gehören unter anderem die der Zitierfähigkeit, der Seitenechtheit, der Druckbarkeit, der Lesbarkeit im Internet (Hypermediafähigkeit), der Archivierfähigkeit, der Recherchierbarkeit sowie der leichten Konvertierbarkeit der Dokumente. Anhand dieser Aufzählung wird schnell klar, dass sich alle diese Anforderungen nicht durch ein einziges Dokumentenformat realisieren lassen. Daher empfiehlt die Arbeitsgruppe, eine Unterscheidung in Erstellungs- bzw. Lieferformate, Präsentationsformate, Retrievalformate und Archivierungsformate vorzunehmen und in diesen Bereichen möglicherweise verschiedene, jeweils prädestinierte Formate zu verwenden. Außerdem werden eine Reihe von Konvertierungswerkzeugen vorgestellt.

Danach wird noch einmal sehr intensiv die Nutzung von Metadaten beschrieben. Die Empfehlungen konzentrieren sich dabei auf Metadaten nach *Dublin Core* [L-DC] und beschreibt den Einsatz dieses Metadatenformates in verschiedenen Formen, so zum Beispiel die Nutzung von *Dublin Core* innerhalb von HTML 4.0 Tags (META-Tags) nach ISO/IEC 11179 und die Abbildung auf das MAB-Format (Maschinelles Austauschformat für Bibliotheken). Interessant ist aber vor allem der Ausblick auf eine XML-basierte Darstellungsform von *Dublin Core* - Metadaten. In diesem Fall handelt es sich um das XML-basierte Datenmodell „DLmeta“, das für die Integration aller Mediengattungen im Bereich des Bibliotheksservice-Zentrums Baden-Württemberg entwickelt wurde. Für die Zukunft ist es absolut wünschenswert, *Dublin Core* Metadaten generell XML-basiert vorzuhalten; die Verwendung eines gemeinsamen Schemas[18] würde den Aufwand des Datenaustauschs zwischen den verschiedenen Institutionen

[18]Leider verwendet „DLmeta“ noch eine Document Type Definition (DTD), diese sollte in Zukunft durch ein XML-SCHEMA[L-SC] abgelöst werden, da durch ein Schema wesentlich genauere Einschränkungen vor allem hinsichtlich korrekter Datentypen bzw. Wertebereiche gemacht werden können.

auf ein Minimum reduzieren und die Etablierung von übergeordneten Protokollen für dezentrale Archive wie z. Bsp. das OAI-Protokoll wesentlich erleichtern.

Im anschließenden Abschnitt wird noch einmal auf die Recherche bzw. Retrieval eingegangen, wobei der Fokus auf einer übergeordneten Metasuche über mehreren dezentralen Archiven liegt. Erstaunlicherweise wird in diesem Zusammenhang zwar kurz auf die Architektur der HARVEST-Software eingegangen und sogar das CARMEN-Projekt von Global Info erwähnt, allerdings verliert das Dokument kein Wort zum OAI-Protokoll oder zur *Open Archives Initiative* im Allgemeinen[19]. Im Folgenden greifen die Empfehlungen noch einmal sehr detailliert das Thema Sicherheitsaspekte auf. Es wird darauf hingewiesen, dass innerhalb des DFN-Vereins damit begonnen wurde, eine Public Key Infrastructure (PKI) speziell für Universitäten und Hochschulen aufzubauen. Ziel ist es, dass jede Hochschule bzw. Universität eine eigene Zertifierungsinstanz (CA) aufbaut, so dass die Identität jedes Nutzers bzw. Computers durch ein digitales Zertifikat bestätigt werden kann. Voraussetzung dafür ist die Aufrüstung der Dokumentenserver und Webserver auf sichere Kommunikationsprotokolle (z. Bsp. SSL) zur verschlüsselten Übertragung von Inhalten.

Der Bereich Langzeitarchivierung wird in den Empfehlungen besonders ausführlich behandelt, da es hier nach wie vor viele Problemstellungen gibt, die auch schon im Abschnitt 2.4 auf Seite 36 angesprochen wurden. Zu den von der Arbeitsgruppe angesprochenen Problemfeldern zählen die Hardwarekomponenten, die Metadaten für die Archivierung, die Speicherung der Daten sowie Geschäftsmodelle für die Steuerung und Überwachung der Archivierungsabläufe. Zum Themenbereich Hardwarekomponenten und -konzepte hat das NEDLIB-Projekt der Europäischen Kommission die Fragen der Langzeitarchivierung vor allem aus dem Blickwinkel der beteiligten europäischen Nationalbibliotheken untersucht. Zu den Ergebnissen dieses Projektes gehören u.a. ein generisches Funktionsmodell für die Archivierung digitaler Publikationen, Strategien für die Langzeitverfügbarkeit digitaler Objekte sowie ein Datenmodell für langzeitarchivierungsrelevante Metadaten. Diese Erkenntnisse sollen an den Nationalbibliotheken - so auch an der Deutschen Bibliothek (DDB) - umgesetzt werden. Aber auch für größere lokale Archive anderer Institutionen lohnt sich ein Blick auf diese Ergebnisse. Für die Archivierungsmetadaten empfiehlt die Arbeitsgruppe das Referenzmodell *„Open Archival Information System“* (OAIS)[L-OAa].

[19]Nach Meinung des Autors ist das schlichtweg unverständlich. Eine Initiative, die bereits über ein halbes Jahr zur Unterstützung der OAI aufgerufen hat, sollte in der Lage sein, in einem Empfehlungspapier (immerhin vom Mai 2001 datiert) diese Initiative bzw. deren Protokoll wenigstens zu erwähnen. Letztendlich sollen ja gerade diese Empfehlungen und die Beachtung von Standards als Grundlage und Vorbereitung für den Einsatz des OAI-Protokolls dienen.

2.4.2.3. Einschätzung und Wertung

Leider werden die Empfehlungen der DINI-Arbeitsgruppe *„Elektronisches Publizieren“* in ihrer vorliegenden, noch unvollständigen Version[20] nach Meinung des Autors dem Anspruch, Empfehlungen für den Umgang mit elektronischen Hochschulpublikationen zu geben, nur bedingt gerecht. Die Arbeitsgruppe muss sich die Frage gefallen lassen, welchem Zweck ihr Dokument letztendlich dienen soll. Soll es eine wissenschaftlich fundierte Betrachtung aller in den Prozess des elektronischen Publizierens involvierten Technologien liefern, oder sollen interessierten Institutionen bzw. Personen praktische Hilfestellungen und Empfehlungen an die Hand gegeben werden, so dass für sie die Barriere zum Aufbau eines OAI-konformen Archives möglichst gering gehalten wird? Wie man die Frage auch beantwortet, beiden Anforderungen kann das Dokument in der vorliegenden Version nicht gerecht werden. Für den ersten Fall ist es an verschiedenen Stellen zu ungenau und fachlich nicht fundiert genug, für den zweiten Fall fehlt es an der nötigen Praxisorientierung und an einfachen, überschaubaren Beispielen, zudem ist das Dokument in diesem Fall viel zu umfangreich und unausgewogen. Vielleicht wäre es eine Lösung, zwei Versionen des Dokuments zu erstellen: eine Version, die umfangreich und sehr detailliert alle zugrundeliegenden Technologien beschreibt, und eine zweite Version, die als kurzes, pointiertes Extrakt der ersten Version mit vielen hilfreichen Beispielen versehen ist und als echtes „Empfehlungspapier“ bzw. Leitfaden für die Praktiker dient. Neben diesen Kritikpunkten enthalten die betrachteten Empfehlungen aber auch viele hilfreiche Informationen, die auch für den Aufbau des HSSS von Interesse sind. Auf diese Punkte wird im Abschnitt 2.4.5 auf Seite 58 näher eingegangen.

2.4.3. Dissertationen Online

Das Projekt *„Dissertationen Online“* ist aus einer Initiative der *„IuK-Kommission der deutschen wissenschaftlichen Fachgesellschaften“*[L-IUK] hervorgegangen und wurde von 1998 bis 2000 von der *„Deutschen Forschungsgemeinschaft“*[L-DF] gefördert. Dabei untergliederte sich das Projekt in 8 verschiedene Teilprojekte, die sich unter anderem mit den Themen Metadaten, Formate, Retrieval, Multimedia sowie Beratung und Unterstützung auseinandersetzten. Seit dem Ende der Förderung sollen die an dem Projekt beteiligten Institutionen durch die *„Koordinierungsstelle DissOnline“* unter Führung der *„Deutschen Bibliothek“* beraten und koordiniert werden. Die Ergebnisse

[20] Stand: Mai 2001

des Projektes werden auf einem speziellen Server[L-Di] präsentiert bzw. sind auch als CD-ROM erhältlich.

Auf diesem Server wurde eine Fülle von Wissen und Informationen zusammengetragen und grob in die Bereiche „Bibliotheken“, „Doktoranden“, „Fachbereiche“ und „Technik“ untergliedert. Im Bereich „Bibliotheken“ sind vor allem allgemeine Informationen sowie Informationen über rechtliche Aspekte, Geschäftsgänge sowie zu Schulungen und Formaten verfügbar. Diese Informationen sind in erster Linie für Bibliotheken bzw. Universitätsbibliotheken gedacht. Die Kategorie „Fachbereiche“ konzentriert sich vor allem auf rechtliche Aspekte im Bereich der Promotionsordnung sowie auf Arbeitsabläufe innerhalb von Fachbereichen an universitären Einrichtungen. Im Bereich „Doktoranden“ finden speziell die Autoren wissenschaftlicher Publikationen Informationen über rechtliche Fragen, Dokumentenformate, Multimediaformate und Metadaten. Der Bereich „Technik“ wendet sich insbesondere an Universitätsbibliotheken bzw. Rechenzentren, die als Anbieter von Diensten im Bereich elektronischer Publikationen fungieren oder solches planen. Hier finden sich Informationen zu den Themen Geschäftsgänge, Dokumentenformate, Metadaten, Recherche und Sicherheit. Die Präsentation der Informationen ist sehr gelungen und wirkt äußerst durchdacht.

In Hinblick auf den HSSS ist der Bereich „Technik“ von besonderem Interesse. Viele der hier verfügbaren Informationen finden sich in etwas gekürzter Form in dem bereits beschriebenen Empfehlungen der DINI-Arbeitsgruppe *„Elektronisches Publizieren“* und sollen aus diesem Grund nicht noch einmal betrachtet werden. Stattdessen wird im folgenden Abschnitt ein für die Zukunft sehr wichtiger Teil im Geschäftsgang elektronischer Publikationen betrachtet: die Verwendung spezieller Formatvorlagen, die es letztendlich erlauben sollen, Metadaten automatisiert generieren zu können. Dies kann erreicht werden, indem das Dokument in einer Auszeichnungssprache vorliegt, die sowohl eine strukturelle, layoutorientierte und inhaltsbezogene Beschreibung desselben erlaubt.

2.4.3.1. Dissertation Markup Language (DiML)

Die *„Dissertation Markup Language“* (DiML) wurde im Rahmen des Projektes *„Dissertationen Online“* an der Humboldt-Universität zu Berlin von Mitarbeitern des dortigen Rechenzentrums entwickelt. Hierbei handelt es sich um eine *Document Type De-*

finition (DTD) auf der Basis von SGML/XML speziell für Doktorarbeiten[21][SMG00]. Da die wenigsten Autoren direkt in XML bzw. SGML schreiben, sondern stattdessen Standardtextverarbeitungssysteme wie z. Bsp. MS Word oder Textsatz-orientierte Software wie Latex oder PDFTex nutzen, müssen für diese Applikationen Formatvorlagen und Plugins existieren, die eine Umwandlung des jeweiligen Formats in das DiML-konforme Zielformat erlauben. Solche Formatvorlagen und Plugins wurden ebenfalls vom Rechenzentrum der Humboldt-Universität zu Berlin entwickelt bzw. getestet und angepasst.

Am Beispiel einer in Word geschriebenen Dissertation lässt sich der gesamte Arbeitsablauf gut beschreiben. Hat sich der Doktorand entschieden, seine Dissertation so zu verfassen, dass sie später in das DiML-Format überführt werden kann, benötigt er als erstes eine Formatvorlage für sein Textverarbeitungssystem (in diesem Fall MS Word). Diese Formatvorlage ist notwendig, um dem originalen Dokumentenformat gewisse Formatanweisungen hinzuzufügen, aus denen später die entsprechenden DiML-Informationen gewonnen werden[22]. Unter Zuhilfenahme dieser Formatvorlage erstellt der Autor nun seine Dissertation und lässt sie am Ende in elektronischer Form der entsprechenden Institution (z. Bsp. Rechenzentrum) zukommen. Hier erfolgt der eigentliche Konvertierungsvorgang. Dazu wird das Dokument als erstes auf die korrekte Nutzung der Formatvorlage überprüft und anschließend eine zusätzliche Formatvorlage (Druckformatvorlage) geladen. Danach wird das Dokument mit Hilfe des Konverters *„Microsoft SGML Author for Word“* und einer speziellen DiML-Konvertierungsdatei nach SGML (DiML) konvertiert. Daraufhin wird die entstandene DiML-Datei durch verschiedene Perl-Skripte nachbearbeitet, unter Umständen ist auch eine manuelle Nachbearbeitung und Fehlerbeseitigung erforderlich. Die am Ende dieser Bearbeitungskette entstandene, der DiML-DTD entsprechende SGML-Datei dient nun als Ausgangsformat für die Generierung weiterer Zielformate, wie zum Beispiel HTML oder PDF.

Die DiML-Struktur lässt sich grob in die Hauptbestandteile **<front>** (Deckblatt), **<body>** (Textkörper) und **<back>** (Anhänge) einteilen. Dabei beinhaltet das Deckblatt alle Metadateninformationen, die über die Dissertation selber festgehalten werden. **Listing 3** zeigt beispielhaft ein Deckblatt in DiML. Der Textkörper

[21] Die DiML-DTD liegt bereits in der Version 2.0[L-Di] vor.

[22] So enthält die Formatvorlage u.a. Dokumentauszeichnungen wie Überschriften, Listen, Tabellen usw. Wird in der Formatvorlage beispielsweise „Überschrift“ gewählt, so erhält der entsprechend folgende Text im Originalformat z. Bsp. die zusätzliche Formatanweisung „diml_headline“. Aus dieser zusätzlichen Formatanweisung kann dann bei der Konvertierung in das DiML-Format z. Bsp. das Fragment „...<head>...text...</head>...“ entstehen (stark vereinfachtes, fiktives Beispiel).

besteht aus dem eigentlichen Inhalt des Dokuments und ist vom Verfasser in mehreren Ebenen gliederbar. Gliederungsstufen sind Kapitel (**<chapter>**), Unterkapitel (**<section>**), Abschnitt (**<subsection>**), Unterabschnitt (**<block>**), Unterunterabschnitt (**<subblock>**) und Part (**<part>**).

```
<front>
<school>
Institut für Klinische Psychologie</p>
der Fachrichtung Psychologie</p>
der Fakultät Mathematik und Naturwissenschaften</p>
der Technische Universität Dresden</p>
</school>
<submission>Dissertation</submission>
<title>Unerwünschte Gedanken bei Angststörungen</title>
<degree>zur Erlangung des akademischen Grades<br/>
doctor rerum naturalium (Dr. rer. nat.)
</degree>
<major>vorgelegt der Fakultät Mathematik und Naturwissenschaften<br/>
der Technischen Universität Berlin
</major>
<author>von
<given>Lydia Birgit</given> <surname>Fehm</surname>
<suffix>06.06.1966 geboren in ...</suffix>
</author>
<dean>Dekan: Prof. Dr. Mustermann</dean>
<approvalls>
<name>Prof. Dr. B. Tuschen−Caffier</name>
<name>Prof. Dr. med. J. Margraf</name>
</approvalls>
<date type="2">eingereicht: 02.07.1999</date>
<date type="2">Datum der Promotion: 19.12.1999</date>
<!−− german −−>
<keywords language="de">
<keyword>Angststörungen</keyword>
...
</keywords>
<!−− english −−>
<keywords language="en">
<keyword>anxiety disorders</keyword>
...
</keywords>
<abstract language="de">
<p>Die vorliegende Arbeit beschäftigt sich mit der Diagnostik unerwünschter ...</p>
</abstract>
<abstract language="en">
<p>The work conducted aimed at evaluating instruments investigating ...</p>
</abstract>
</front>
```

***Listing 3** : Struktur eines Deckblattes in DiML*

Überschriften werden in jeder Gliederungsebene durch **<head>** ausgezeichnet. In den einzelnen Gliederungsbestandteilen ist jeweils die selbe Binnenstruktur verfügbar, bestehend aus **<p>** (Absatz), **<citation>** (bibliographische Referenz), **<table>** (Tabellenstrukturen) und **<mm>** (Multimediaobjekte). Außerdem sind mathematische Formeln, Fußnoten, Endnoten und Links sowie Listen beschreibbar. Auch für

spezielle Auszeichnungen bzw. Formatierungen oder Hervorhebungen innerhalb des Textes existieren Tags zur entsprechenden Beschreibung. Die Anhänge stellen den dritten Hauptbestandteil eines DiML-Dokuments dar und bestehen aus den Elementen **<bibliography>** (Bibliographie), **<abbreviation>** (Abkürzungsverzeichnis), **<vita>** (Lebenslauf), **<declaration>** (Selbstständigkeitserklärung) und **<acknowledgements>** (Danksagungen) sowie frei definierbaren Anhängen.

Die Vorteile, die sich aus der Verwendung einer solchen Auszeichnungssprache ergeben, liegen klar auf der Hand:

- Es steht ein Archivierungsformat zur Verfügung, welches im Vergleich zu anderen Formaten wesentlich einfacher über Hard- und Softwaregrenzen hinaus austauschbar ist[23].
- Durch die Aufnahme der Dokument-Metadaten in den Deckblatt-Teil des DiML-Dokuments lassen sich genau diese Metadaten automatisiert gewinnen. Voraussetzung ist, dass nur Dokumente mit vollständigen Metadaten (durch die DTD zu definieren) vom Autoren abgegeben werden, was sehr einfach durch einen entsprechenden Parser festgestellt werden kann.
- Durch die zusätzlichen strukturellen und inhaltsbasierten Informationen sind erheblich erweiterte Recherchemöglichkeiten im Volltext gegeben.
- Das Format eignet sich ausgezeichnet als Ausgangsformat zur Generierung weiterer Zielformate wie etwa HTML oder PDF. Insbesondere ist die Wahrscheinlichkeit, dass es sich auch in Zukunft zur Generierung noch unbekannter Dokumentenformate eignet, im Vergleich zu herkömmlichen Dokumentenformaten wesentlich höher.

Ein mögliches Problem dieses Ansatzes ist allerdings die Generierung des Präsentationsformates aus der Auszeichnungssprache. Die Frage ist, ob wirklich sichergestellt werden kann, ob das entstehenden Layout dem ursprünglichen Layout des Autors exakt gleicht, was von der Mehrheit der Autoren erwartet wird. Falls dies nicht garantiert werden kann, steht zu befürchten, dass ein solcher Ansatz langfristig nicht akzeptiert wird. Eine Kompromisslösung wäre unter Umständen, eine zweigleisige Strategie zu verfolgen, indem das Druckformat respektive Präsentationsformat auf herkömmliche Weise aus dem Original erstellt wird (ohne strukturelle oder semantische Informationen zu nutzen) und die Auszeichnungssprache nur für Formate verwendet wird, die

[23] Gerade auch im Hinblick auf die im Abschnitt 2.4 auf Seite 36 getroffenen Feststellungen in Bezug auf Langzeitarchivierung ist diese Tatsache von großer Bedeutung.

davon auch tatsächlich profitieren können[24]. Dies würde Sinn machen, wenn man bedenkt, dass ein Präsentationsformat im Sinne eines Druckformats die zusätzlichen semantischen bzw. strukturellen Kontextinformationen eigentlich nicht benötigt, da seine einzige Aufgabe darin besteht, das Layout des Dokuments so exakt wie möglich zu kodieren.

2.4.4. Subjektive Einschätzung der betrachteten Initiativen und Entwicklungen

Leider fällt das Resümee der betrachteten Initiativen im Umfeld des Diplomthemas nicht sehr positiv aus. Generell mangelt es gerade für Neulinge in diesem Gebiet an Transparenz. So gibt es eine Vielzahl sich thematisch überschneidender Initiativen mit teilweise alternativen und konkurrierenden Ansätzen. Eine Vielzahl der verwendeten Technologien ist zudem veraltet und schlecht dokumentiert. Erster Kritikpunkt ist daher die äußerst schlechte Koordination. Nach Meinung des Autors fehlt es einfach an einer übergeordnet verantwortlichen Organisation, die für die nationale Koordinierung aller Aktivitäten legitimiert ist. Nur dadurch würden sich überschneidende Forschungsprojekte und inkompatible Archive verhindern bzw. gemeinsame Zielsetzungen und Protokolle wirksam und vor allem wesentlich schneller umsetzen lassen. Der nächste Kritikpunkt betrifft die vorhandenen Hilfestellungen. In vielen Fällen sind sie nicht praxisorientiert genug, um als erfolgreiche Anleitung für eine potentielle Neuentwicklung eines Archives genutzt werden zu können. Hier wäre weniger oft mehr. Für eine Institution, die sich entscheidet, ein digitales Archiv aufzubauen, sind in erster Linie nicht alle nur möglichen technischen Gesichtspunkte interessant, sondern kurze, exakte Aussagen zu den wichtigsten Punkten, die unbedingt beachtet werden sollten. Auch das ansonsten sehr gut gestaltete Informationsangebot von *„Dissertationen Online“* würde durch eine gestraffte Zusammenfassung aller existentiellen Anforderungen an Übersichtlichkeit gewinnen. Ein besonders positiver Umstand ist die OAI und insbesondere deren OAI-Protokoll. Hierbei handelt es sich, wie bereits gezeigt wurde, um ein schlankes, aber wirkungsvolles und in der Praxis leicht umsetzbares Protokoll, welches mittelfristig dazu führen könnte, den Suchraum in bezug auf wissenschaftliche Publikationen von einzelnen Archiven auf eine beliebige Menge lose gekoppelter, dezentraler Archive auszuweiten. Leider ergibt sich aus diesem Positivum der dritte Kritikpunkt, der nach Meinung des Autors in der zu langsamen Umsetzung aktueller Entwicklungen besteht. So sind viele Entwicklungen und Dokumentationen

[24] also andere Auszeichnungsformate o.ä., die zum Beispiel für die Archivierung oder interaktive Online-Präsentationen usw. genutzt werden können

technisch nicht auf dem Stand der Zeit. Bei dem betrachteten Gebiet handelt es sich um ein Gebiet, welches besonders intensiv technischen Veränderungen unterworfen ist. Das verlangt entsprechend schnelle Reaktionen und damit insbesondere die schnelle Umsetzung aktueller Standards. Die Realität sieht leider anders aus. So hat zum Beispiel die DINI bereits im September 2000 in ihrem Appell dazu aufgerufen, das OAI-Protokoll zu unterstützen. Gleichzeitig erklärte sich die DINI dazu bereit, diesen Prozess unterstützend zu begleiten und zu fördern. Eine erste Informationsveranstaltung fand aber erst ein dreiviertel Jahr später am 18. Juni 2001 statt. So ist es nicht verwunderlich, dass zum jetzigen Zeitpunkt - nach fast einem Jahr - erst etwa vier bis fünf Archive mit OAI-Protokoll-Implementierung existieren. Von einer breiten Durchsetzung dieses Protokolls kann also keine Rede sein. Wie ließen sich diese Mängel beheben? Folgende Punkte könnten nach Einschätzung des Autors zu einer Besserung der Situation führen:

- Als erste Voraussetzung muss sich der Bund (genauer: das Bundesforschungsministerium) entscheiden, wie wissenschaftliche Publikationen rechtlich und politisch gehandhabt werden. Nach der Überzeugung des Autors kann die Entscheidung nur lauten: Wissenschaftliche Publikationen sind im wesentlichen durch Steuergelder finanzierte, immaterielle Werte. Aus diesem Grund sind sie „Public Domain“, müssen also der gesamten Öffentlichkeit und insbesondere der Wissenschaft selbst ohne größere Zusatzkosten zugänglich sein.

- Als logische Konsequenz daraus folgt zwingend, dass das Bundesforschungsministerium eine Organisation ins Leben rufen muss, welche die folgenden Anforderungen durchsetzen kann und mit entsprechenden Legitimationen ausgestattet ist. Diese Organisation muss zum Ziel haben, die nationale Infrastruktur im Bereich digitaler wissenschaftlicher Publikationen und 'e-print'-Services so auszubauen, dass diese Dienste für wissenschaftliche Einrichtungen zum Selbstkostenpreis verfügbar sind. Zu diesem Zweck tritt die Organisation mit Legitimationen auf, die es ihr erlauben, alle Tätigkeiten in diesem Bereich national zu koordinieren. Sie erarbeitet klare Mindestanforderungen, die notwendig sind, um als „wissenschaftliches Archiv“ zertifiziert werden zu können. Nur Archive, die dieses Zertifikat besitzen oder es durch Weiterentwicklungen ausdrücklich erlangen wollen, werden gefördert. Neben diesen nationalen Aufgaben beobachtet die Organisation die internationale Entwicklung und nimmt an entsprechenden Veranstaltungen teil, wobei der große Vorteil besteht, dass sie Deutschland als Ganzes vertreten kann. Neben technischen Fragen gehören hier vor allem auch juristische und konzeptionelle Fragen zu den Aufgabengebieten dieser Organisation. Andere regionale Organisationen haben nach wie vor ihre Berechtigung, allerdings sind einzig und allein die Richtlinien der nationalen Organisation für alle wissenschaftlichen Archive bindend.

- Zu den wichtigsten Aufgaben der Organisation gehört es, den Aufbau von dezentralen, aber zueinander kompatiblen, langfristig verfügbaren Archiven zu koordinieren. Dazu ist es zwingend erforderlich, eine Reihe von Mindestanforderungen zu spezifizieren, die von allen Archiven unbedingt zu realisieren sind. Bei heutigem Stand der Technik würden folgende Anforderungen dazu zählen:

  - Das OAI-Protokoll ist vom Archiv zu implementieren, d.h. jedes Archiv ist in der Lage, als *Data Provider* aufzutreten.
  - Es ist von allen Archiven zumindest das Metadatenformat nach Dublin Core zu unterstützen.
  - Jede Ressource des Archivs ist langfristig stabil erreichbar, d.h. es ist eine Indirektion zu realisieren, die dazu führt, dass die Ressource selbst bei einer Änderung der physischen Persistenz immer unter dem selben Identifier erreichbar ist.
  - Jedes Archiv garantiert technisch für eine Mindestverfügbarkeit in einem tolerierbaren Bereich (ca. 99%).

  Zusätzlich zu diesen Mindestanforderungen sollte die Organisation weitere, optionale Anforderungen spezifizieren. Vielleicht könnte - wie bei einigen Protokollen üblich - der Grad der Übereinstimmung der Implementation mit den Anforderungen in verschiedenen „Leveln“ angegeben werden. Je höher das Level ist, das ein Archiv implementiert, desto größer sollte die Unterstützung und Förderung durch die Organisation sein.

- Alle Archive sind durch die Organisation durch entsprechende Tests der Implementation zu zertifizieren. Die Archive sollten aber auch nach der Zertifizierung in unregelmäßigen Abständen - möglichst automatisiert - überprüft und getestet werden.

- Die Organisation selber sollte Dokumentationen und Hilfestellungen zur Verfügung stellen. Außerdem sollte sie eine Referenzimplementation eines „wissenschaftlichen Archivs“ öffentlich machen, welches als Beispiel oder als Grundlage einer eigenenen Entwicklung für beliebige Institutionen dienen kann. Diese Referenzimplementation sollte auch optionale Bereiche abdecken, vor allem aber so weit wie möglich moderne, zukunftsträchtige Technologien verwenden. Beim derzeitigen Stand der Technik wären nach Ansicht des Autors folgende Anforderungen an eine solche Referenzimplementation sinnvoll:

  - Es sollte eine moderne Sprache und eine moderne Architektur gewählt werden. Ideal wäre Java in Verbindung mit Server-Technologien wie Servlets bzw. JSP's, vielleicht auch die Implementation als J2EE-Anwendung.

– Es sollten, so weit es möglich ist, Open Source Technologien verwendet werden. Die Referenzimplementation sollte deshalb den Apache Webserver und gegebenenfalls den Tomcat Servlet-Container bzw. einen EJB-Container oder Enterprise Application Server wie etwa JBoss verwenden.

– XML-Technologien sollten so viel wie möglich Verwendung finden. In die oben angesprochene Kombination von Java und Open Source - Komponenten der Apache Software Foundation (ASF) lassen sich XML-Technologien geradezu ideal einpassen[25].

2.4.5. Realisierungsmöglichkeiten für den HSSS

In diesem Abschnitt sollen - basierend auf der Aufgabenstellung sowie den Anforderungen von DINI und *„Dissertationen Online“* - die wichtigsten Anforderungen an den HSSS spezifiziert werden. Dabei können nicht alle von den genannten Institutionen vorgeschlagenen Anforderungen in der während der Diplomarbeit zu erstellenden ersten Ausbaustufe des HSSS berücksichtigt werden. Aus diesem Grund muss eine begründete Auswahl getroffen werden, wobei die Kosten-Nutzen-Relation zwischen dem erwarteten Implementierungsaufwand und der Priorität der jeweiligen Anforderung zu beachten ist.

Ein großer Teil der geforderten Merkmale eines Dokumentenservers konnte schon in der durch die Belegarbeit entstandenen Version des HSSS realisiert werden[Schb00]. Dazu zählen:

- Ein vorläufiges, noch zu erweiterndes Geschäftsmodell
- Die Unterstützung des Metadatenformats nach *Dublin Core*
- Die Spezifikation von Metadatensätzen für die zu unterstützenden Dokumentenarten
- Eine Analyse der in Frage kommenden Technologien für die Realisierung eines Dokumentenservers

[25] Auch der Bereich der XML-Technologien lässt sich mit Open Source Software ausgezeichnet abdecken. So bietet die ASF u.a. einen hervorragenden XML-Parser (Xerxes), eine XSLT-Engine (Xalan) sowie Implementationen diverser XML-Protokolle (z. Bsp. SOAP).

- Eine begründete Technologieentscheidung sowie eine lauffähige Kombination von Softwaresystemen (Web-Server, Servlet-Container etc.), die als Grundlage für die zu erstellenden Anwendungen dienen
- Eine Anwendung, die es Autoren erlaubt, Metadaten zu ihren Dokumenten zu erstellen und diese an die SLUB zu senden
- Eine partiell realisierte Anwendung, die die Auswertung und Bearbeitung der Metadaten erlaubt

Aus der Aufgabenstellung der Diplomarbeit bzw. aus den Anforderungen respektive Empfehlungen von DINI und *„Dissertationen Online“* sowie aus eigenen Überlegungen ergeben sich folgende weitere Merkmale, die der HSSS unterstützen wird:

- Das schon vorhandene Geschäftsmodell wird erweitert und insbesondere in den Punkten interne und externe Kommunikation verbessert und abgestimmt. Die Praxistauglichkeit und Funktionstüchtigkeit der spezifizierten Arbeitsabläufe wird durch Tests nachgewiesen.
- Die Metadaten zu den Dokumenten werden persistent abgelegt. Die Persistenz wird dabei durch ein noch zu bestimmendes DBMS realisiert.
- Durch die Verwendung einer Indirektion (stabile URL) wird die langfristige Verfügbarkeit einmal publizierter Dokumente sichergestellt.
- Die Metadatensätze werden (bis auf die endgültige Speicherung im DBMS) im XML-Format vorgehalten. Dies ermöglicht einen flexibleren Umgang mit den Metadaten und erleichtert die Übertragung bzw. den Export an andere Institutionen.
- Alle Anwendungen werden durch die Integration von XML-basierten Konfigurationsdateien äußerst flexibel und von außen veränderbar gestaltet.
- Die Anwendung zur Bearbeitung und Administration der Metadaten wird vervollständigt und unterstützt die spezifizierten Arbeitsabläufe bis hin zur persistenten Speicherung in das DBMS so perfekt wie möglich.
- Es ist eine Anwendung verfügbar, die die Suche in den persistenten Metadaten erlaubt und nach Möglichkeit auch eine Volltextsuche unterstützt.
- Die Spezifikationen der OAI werden vom HSSS dergestalt implementiert, dass der Hochschulschriftenserver als *Data Provider* auftreten kann und somit die

Inhalte international verfügbar sind. Um diese Anforderung erfüllen zu können, müssen gegebenenfalls an anderen (unwichtigeren) Stellen Einschränkungen hingenommen werden.

Damit sind fast alle wesentlichen Anforderungen an einen Hochschulschriftenserver erfüllt. Insbesondere sind durch die Unterstützung des Dublin Core Metadatenformats sowie die Verwendung von XML-Technologie die Voraussetzungen für die Implementation des OAI-Protokolls und damit für die internationale Verfügbarkeit des Dokumentenservers gegeben. Die Implementation dieses Protokolls gehört zu einigen erweiterten Anforderungen, die für den Hochschulschriftenserver mit Sicherheit von großem Vorteil wären und damit auch eine gewisse Priorität haben, die aber nicht zu den existentiellen Merkmalen gehören und damit nur in Abhängigkeit der zur Verfügung stehenden Zeit realisiert werden können. Insofern kann für die Realisierung während der Diplomarbeit nicht garantiert werden. Zu diesen Anforderungen zählen auch die Integration von Kommunikationsprotokollen wie etwa E-Mail bzw. FTP und administrativen Werkzeugen[26] in die Anwendung. Die folgenden Anforderungen an einen Hochschulschriftenserver sind ebenfalls gerechtfertigt und von hoher Priorität, können aber aufgrund ihres großen Realisierungsaufwands im Rahmen des Diploms nicht angegangen werden. Zu diesen Anforderungen gehören:

- Die Anforderung nach der Einbindung in eine PKI. Dazu ist es notwendig, dass der Web-Server und die Anwendung ein sicheres Protokoll (z. Bsp. SSL) unterstützen (bei HSSS realisiert), aber auch der Aufbau einer eigenen CA, die in der Lage ist, sowohl Nutzer, Rechner, Dokumentenserver und auch Dokumente zu zertifizieren. Das ist mit hohem technischen und zeitlichen Aufwand verbunden und muss einer kommenden Ausbaustufe des HSSS vorbehalten bleiben.
- Die Nutzung der bereits vorgestellten Auszeichnungssprache DiML und der damit zusammenhängenden Werkzeuge brächte einige wesentliche Vorteile mit sich, kann aber insbesondere aufgrund des hohen organisatorischen Aufwands noch nicht im Rahmen des Diploms realisiert werden. Es ist aber unbedingt anzuraten, diese Technologien in einer weiteren Ausbaustufe des HSSS in Zusammenarbeit mit den Fakultäten der TU Dresden einzuführen.

[26] Zum Beispiel eine integrierte Benutzerverwaltung und Bearbeitung der Konfigurationsdateien innerhalb der Administrations-Anwendung etc.

3. Systemarchitektur

Inhalt des Kapitels:

3.1. Geschäftsmodelle und Arbeitsabläufe 62
3.1.1. Geschäftsmodell für Dissertationen 64
3.1.2. Arbeitsablauf für Berichte 71
3.1.3. Einschätzung der Arbeitsabläufe 73
3.2. Architekturübersicht 78
3.2.1. Architekturvision 80
3.2.2. Vorgehensweise, Priorisierung 88
3.3. Architekturdetails 91
3.3.1. Architekturbesonderheiten 92
3.3.1.1. Bearbeitungsnummer 92
3.3.1.2. Konfigurationsstrukturen 94
3.3.2. Architekturdetails auf Server-Ebene 96
3.3.2.1. Technologische Alternativen 97
3.3.2.2. Objektorientierte Analyse und Design 99
3.3.2.3. Architektur- und Entwurfsmuster 105
3.3.3. Architekturdetails auf Client-Ebene 108
3.3.3.1. Einordnung in die Gesamtarchitektur 108
3.3.3.2. Technologische Alternativen 111
3.3.4. Architekturdetails auf Datenbank-Ebene 113
3.3.4.1. Schema- und Datenbankanforderungen 114
3.3.4.2. Datenbank-Evaluation 115
3.3.4.3. Design der Schnittstelle 118

3.1. Geschäftsmodelle und Arbeitsabläufe

Dieser Abschnitt soll alle im Zusammenhang mit dem HSSS stehenden Arbeitsabläufe analysieren und detailliert beschreiben, noch bevor im weiteren Verlauf des Kapitels mit der technischen Analyse und dem Design der Systemarchitektur begonnen wird. Für die Entwicklung von kundenorientierter Software und insbesondere für die spätere Akzeptanz und Effektivität der Anwendungen ist es unabdingbar, die vorhandenen Arbeitsabläufe und Prozesse zu analysieren und nach Möglichkeit zu rationalisieren. Aus diesem ersten Schritt im Lebenszyklus einer Softwareentwicklung sollen die entscheidenden Impulse für den Entwurf und die primären Anforderungen an die Anwendung klar hervorgehen. In dieser Phase ist es wichtig, durch konstruktive Kommunikation mit dem Kunden so viel seines fachlichen Wissens wie möglich in den weiteren Entstehungsprozess der Software einfließen zu lassen. Am Ende sollte sich ein komplettes Geschäftsmodell bzw. eine Reihe von Arbeitsabläufen ergeben, so dass das Verhalten aller mit der Software im Zusammenhang stehenden Prozesse beschrieben wird.

Zu Beginn der Belegarbeit und zum Teil auch noch zu Beginn der Diplomarbeit war bei der SLUB kein explizites Geschäftsmodell zur Behandlung elektronischer Dokumente vorhanden. Solche Dokumente wurden quasi 'nebenbei' auf Disketten und CD-ROM's angenommen, allerdings in keinem geordneten Verfahren. Der Fokus lag einzig und allein auf den traditionellen, papiergebundenen Publikationen, nach denen sich auch sämtliche Arbeitsabläufe ausrichteten. Die zusätzliche Unterstützung von digitalen Dokumenten verlangt aber zum Teil ganz andere Arbeitsabläufe, zudem sind andere Personen respektive Abteilungen involviert. Die besondere Herausforderung war daher die Synchronisation der traditionellen Arbeitsabläufe mit den sich neu ergebenden. Eine zusätzliche Schwierigkeit bestand darin, dass sich das Geschäftsmodell über eine Vielzahl zu beteiligender Personen und Abteilungen sowohl innerhalb als auch außerhalb der SLUB erstreckt. Daher musste das Geschäftsmodell besonders intensiv mit allen Beteiligten besprochen und auf die jeweiligen Bedürfnisse abgestimmt werden.

Die entstandenen Arbeitsabläufe werden getrennt für jede der zwei derzeit vom HSSS unterstützten Dokumentenarten vorgestellt. Zur Beschreibung wird die *Unified Modeling Language* (UML) genutzt, die sich als Standardmodellierungssprache inzwischen weltweit durchgesetzt hat[BRJ99]. Zur Beschreibung der Arbeitsabläufe eignen sich dabei vor allem Diagramme, die in der Lage sind, die dynamischen Aspekte eines Systems zu beschreiben. Daher werden für die Erläuterung des Geschäftsmodells im Folgenden Anwendungsfalldiagramme und Sequenzdiagramme genutzt. Zunächst sol-

len aber die innerhalb der SLUB beteiligten Referate vorgestellt werden. **Tabelle 2** zählt diese auf und erläutert zusätzlich die jeweils wichtigsten Aufgabenbereiche[1].

Referat	Aufgabenbereiche
Referat 3.1	*Referat für Monographien und Sonderformen*
Referat 3.4	*Referat für Pflicht, Tausch und Geschenk* Dieses Referat beinhaltet die Hoschschulschriftenstelle, die den Schriftentausch (Druckexemplare) mit Veröffentlichungen der TU Dresden realisiert sowie die Pflichtabgabestelle, welche in Sachsen erschienene Druckerzeugnisse entgegennimmt (z. Bsp. Berichte)
Referat 4.1 + 4.2	*Referat für Formalerschließung (einschl. Zeitschriften und ZDB sowie „non book materials“)* Diese Referate erschließen die entsprechenden Materialien für den Alphabetischen Katalog, dabei erfolgt für Monographien bzw. Zeitschriften eine Verbund-Katalogisierung nach formalen Gesichtspunkten, für elektronische Dissertationen wie auch für andere Medien erfolgt die „Hochkatalogisierung“ in Zusammenarbeit mit dem SWB.
Referat 5.3	*Referat für Informationsvermittlung und neue Dienstleistungen* Dieses Referat ist unter anderen für die Informationsvermittlung elektronischer Datenbestände wie z. Bsp. CD-ROM-Datenbanken, Online-Datenbanken und elektronischen Zeitschriften sowie für die Entwicklung und Organisation aller Geschäftsabläufe im Bereich des Hochschulschriftenservers HSSS verantwortlich.
Referat 10.* + 11.*	*Zweigbibliotheken und Fachreferate* Diese Referate sollen für den jeweiligen Fachbereich eine(n) Bestandsaufbau, -pflege, -erschließung und -vermittlung realisieren und sind außerdem jeweils für Fachinformationen zuständig.

***Tabelle 2** - Am Geschäftsmodell beteiligte Abteilungen der SLUB*

[1] dabei steht Referat 10.* für alle Referate der Abteilung 10, gleiches gilt für Referat 11.*

3.1.1. Geschäftsmodell für Dissertationen

Aufgrund der Komplexität des gesamten Arbeitsablaufs wird dieser in mehrere chronologisch aufeinanderfolgende Teile zerlegt, wobei die Betrachtung gleichzeitig auf einen bestimmten Akteur fokussiert ist. Am Beginn des Arbeitsablaufs steht der Autor einer Dissertation[2], der sich entschieden hat, diese auf elektronischem Wege weltweit langfristig verfügbar zu machen. **Abbildung 2** zeigt ein Anwendungsfalldiagramm, welches den Kontext des Gesamtsystems speziell aus dem Blickwinkel des Autors beschreibt:

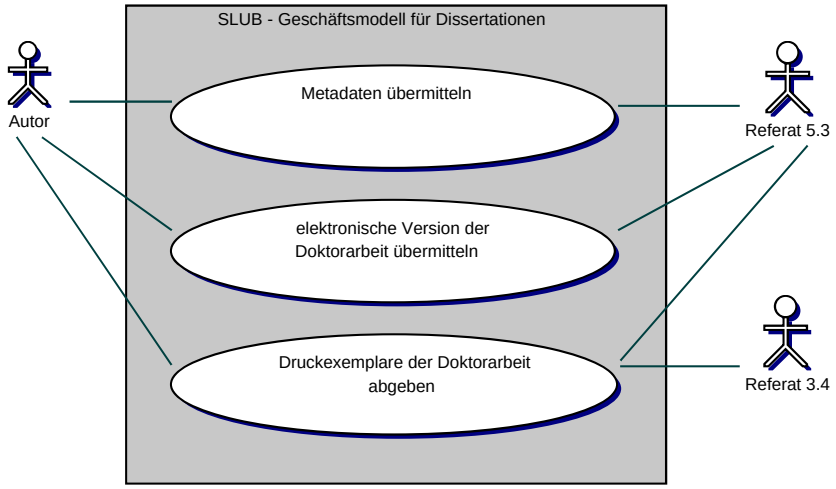

Abbildung 2 *- Anwendungsfalldiagramm: Kontext des Systems für den Autor*

Wie die Abbildung zeigt, lassen sich für den Akteur insgesamt drei Anwendungsfälle ausmachen, bei denen er mit dem Gesamtsystem interagiert. Als erstes wird er über eine vom System zur Verfügung zu stellende Schnittstelle Metadaten übermitteln, die ihn und insbesondere sein Dokument beschreiben. Als nächstes wird der Autor die elektronische Version seiner Dissertation in einem bestimmten Format an das System übermitteln. Welches Format verwendet werden darf und ob die Übermittlung ebenfalls über die Metadatenschnittstelle erfolgt oder getrennt davon, ist dabei auf diesem Abstraktionsniveau nicht von Interesse. Mit den beiden bis hierher beschriebenen Anwendungsfällen steht ebenfalls der Akteur „Referat 5.3“ in Verbindung. Als letztes muss der Autor dem System eine gewisse Anzahl an Druckexemplaren seines Dokuments zur Verfügung stellen. Mit diesem Anwendungsfall steht neben dem Referat 5.3

[2]wobei Dissertation hier und im Folgenden im Sinne der Dokumentenart “Dissertation“ gemeint ist, so dass es sich jeweils um eine Dissertation oder eine Habilitation handeln kann

auch das Referat 3.4 in Verbindung. **Abbildung 3** zeigt das dazugehörige Sequenzdiagramm, welches die beschriebenen Sachverhalte noch detaillierter darstellen kann, insbesondere was deren zeitliche Reihenfolge betrifft.

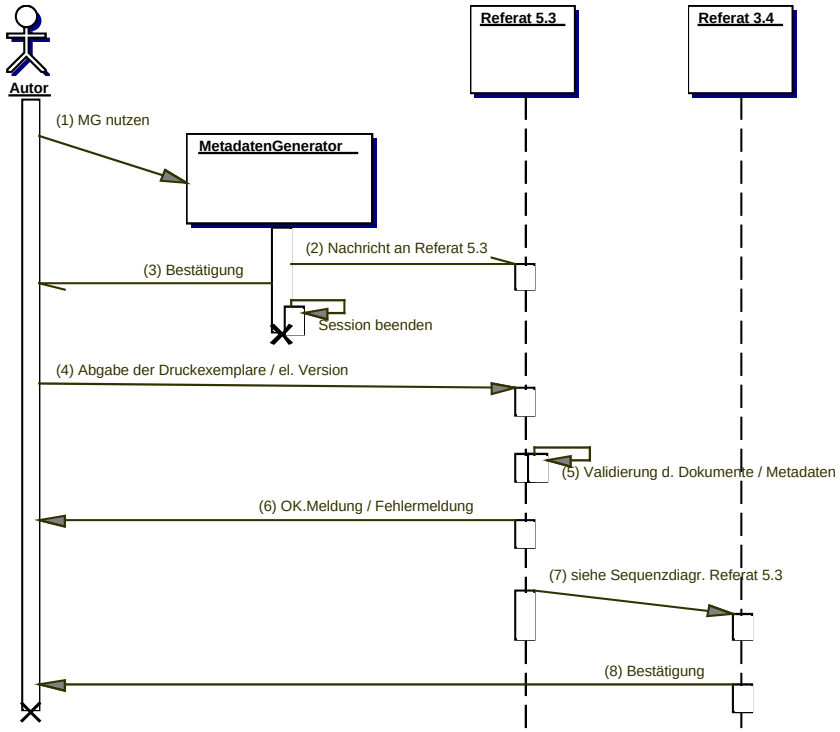

Abbildung 3 *- Sequenzdiagramm: Geschäftsgang in Bezug auf den Autor*

Die Abbildung zeigt den Akteur „Autor“ und die für ihn sichtbaren Instanzen (Objekte) des Systems sowie die zeitliche Reihenfolge aller Nachrichten zwischen diesen Objekten. Als erstes **(1)** 'nutzt' der Autor die vom System zur Verfügung gestellte Schnittstelle zur Übermittlung der Metadaten. Bei dieser Schnittstelle handelt es sich um die im Diplom noch leicht zu modifizierende Version der im Rahmen der Belegarbeit entstandenen Anwendung „*MetadatenGenerator*“. Das Verb „nutzen“ steht dabei für eine Reihe von Interaktionen, die der Autor mit dieser Anwendung vornehmen muss, eine genauere Spezifikation dieser Aktionen wurde bereits in der Belegarbeit[Schb00] vorgenommen und liegt außerhalb des hier betrachteten Abstraktionsgrades. Am Ende dieser Nutzung erhält das Referat 5.3 eine Information über den Eingang eines neuen Metadatensatzes **(2)** und der Autor gegebenenfalls eine Bestätigungsnachricht **(3)** jeweils in Form einer E-Mail. Anschließend gibt der Autor die geforderte Anzahl von Druckexemplaren und die elektronische Version seines Dokuments[3] im Referat 5.3 ab **(4)**. Jetzt werden sowohl die Metadaten als auch die

[3]Dazu sind vom System sowohl die akzeptierten Formate als auch die mögliche Übermittlungsmodi

Dokumente des Autors von Mitarbeitern des Referats 5.3 validiert[4] **(5)**. Im Falle einer erfolgreichen Validierung erhält der Autor eine Bestätigung, andernfalls eine Fehlermeldung, die ihn anweist, die notwendigen Ressourcen (Metadaten, Druckexemplare, digitales Dokument) korrekt zur Verfügung zu stellen **(6)**. Vom Referat 5.3 werden eine Reihe von Aktionen in Verbindung mit dem Referat 3.4 ausgeführt, die in einem noch folgenden, das Referat 5.3 betreffenden Sequenzdiagramm detaillierter dargestellt werden **(7)**. Als letztes erhält der Autor vom Referat 3.4 eine Bestätigung, die er der Prüfungskommission vorlegen muss, um seine Promotionsurkunde zu erhalten **(8)**.

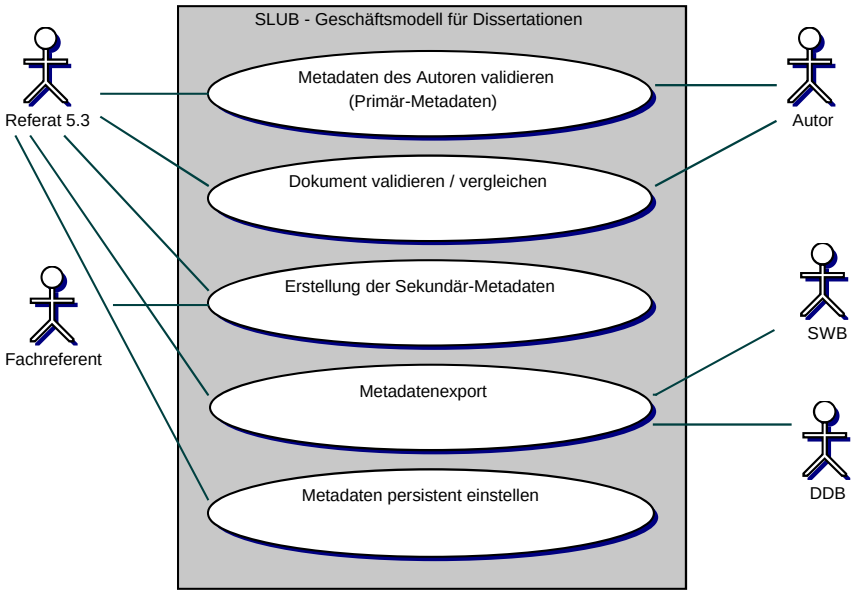

Abbildung 4 *- Anwendungsfalldiagramm: Kontext des Systems für das Referat 5.3*

Abbildung 4 zeigt die in erster Linie mit dem Referat 5.3 verbundenen Anwendungsfälle. Als erstes werden die vom Autor übermittelten Metadaten validiert. Dann werden die vom Autor gelieferten Dokumente (Druckexemplare und elektronische Version) überprüft, indem die Druckexemplare stichprobenartig auf Vollständigkeit und die digitale Version auf formale Übereinstimmung mit der Druckvorlage kontrolliert wird. Bei diesen beiden Anwendungsfällen ist natürlich auch der Autor als Akteur involviert, der gegebenenfalls über Ergebnisse informiert wird. Als nächstes werden die sekundären Metadaten[5] vom entsprechenden Fachreferenten in Zusammenarbeit

festzulegen. HSSS erlaubt die Dokumentenformate PDF, PS und DVI sowie die Übermittlung per E-Mail, die Angabe einer URL oder die Abgabe auf einem von der SLUB lesbaren Datenträger.

[4]Ein Großteil der Validierung ist aber auch schon durch den MetadatenGenerator realisiert worden. Es müssen also nur noch technisch nicht automatisierbare Validierungen durchgeführt werden.

[5]Sekundäre Metadaten beinhalten vor allem bibliothekarische Informationen und Klassifikationen.

mit dem Referat 5.3 erstellt. Der vollständige Metadatensatz wird nun an die Akteure SWB (*Südwestdeutscher Bibliotheksverbund*) und DDB (*Die Deutsche Bibliothek*) exportiert. Als letztes ist der Metadatensatz in die persistente Datenbasis des Gesamtsystems einzustellen. Einen detaillierteren und zeitlich differenzierten Überblick über die beschriebenen Anwendungsfälle gibt wieder das dazugehörige Sequenzdiagramm, dessen erster Teil in **Abbildung 5** zu sehen ist:

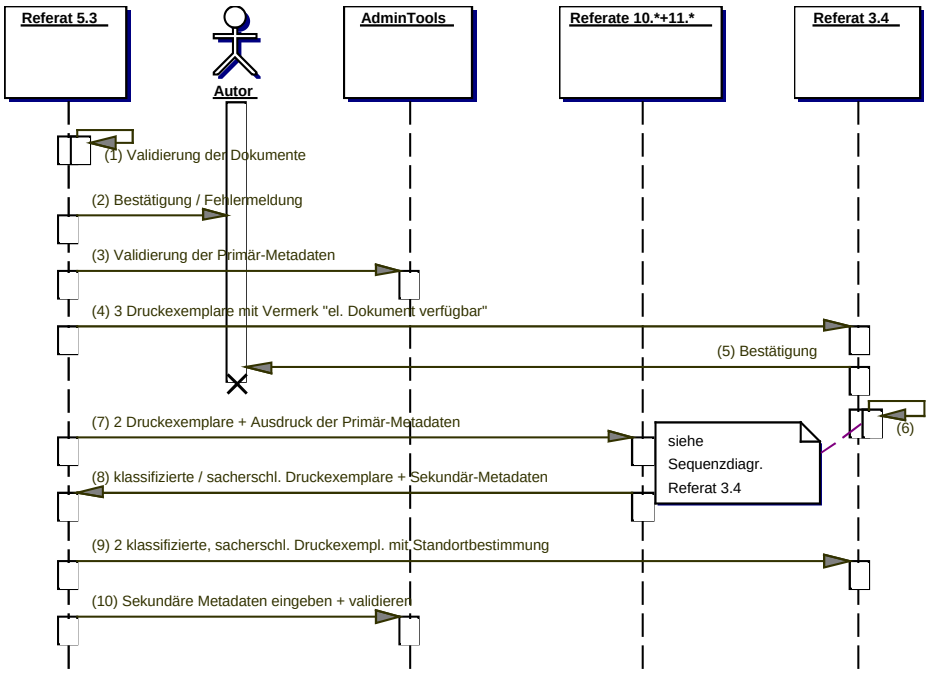

Abbildung 5 *- Sequenzdiagramm:*
Geschäftsgang in Bezug auf das Referat 5.3 - Teil 1

Zunächst werden die Dokumente des Autors validiert **(1)**. Wie bereits erwähnt, schließt das sowohl eine stichprobenartige Kontrolle der Pflichtexemplare (Druckexemplare) als auch den ebenfalls stichprobenartigen Übereinstimmungsvergleich zwischen Druckexemplar und elektronischer Version der Dissertation ein. Im Falle einer erfolgreichen Validierung erhält der Autor eine Bestätigung, ansonsten eine Fehlermeldung **(2)**. Anschließend werden die Primär-Metadaten des Autors validiert **(3)**. Dies geschieht mit Hilfe einer Administrationsanwendung, die hier mit „*AdminTools*“ bezeichnet wurde. Diese Anwendung sollte alle weiteren Schritte des Arbeitsablaufs - soweit möglich - technisch unterstützen. Drei der fünf vom Autor in gedruckter Form abzugebenden Pflichtexemplare seiner Dissertation werden an das Referat 3.4 mit dem Vermerk „Zu diesem Dokument ist in Kürze eine elektronische Version verfügbar“ verschickt **(4)**. Auf diesen Vermerk wird zu einem späteren Zeitpunkt bei der Erläuterung eines weiteren Sequenzdiagramms näher eingegangen. Vom Referat 3.4 erhält

der Autor eine Empfangsbestätigung, die er zum Erhalt seiner Promotionsurkunde benötigt **(5)**. Mit dieser Bestätigung ist der Arbeitsablauf aus Sicht des Autors vorerst beendet. Außerdem führt das Referat 3.4 eine Reihe von Bearbeitungsschritten durch, die in einem gesonderten Diagramm beschrieben werden **(6)**. Als nächstes schickt das Referat 5.3 zwei Druckexemplare sowie eine Zusammenfassung der Primärmetadaten an den jeweiligen Fachreferenten des entsprechenden Referats der Abteilung 11 **(7)**. Gegebenenfalls kann diese Aktion auch in Verbindung mit der Administrationsanwendung erfolgen. Der Fachreferent nimmt die entsprechende Sacherschließung und Standortbestimmung für die beiden in der SLUB verbleibenden Druckexemplare vor und erstellt die sekundären Metadaten[6] für die elektronische Version des Dokuments. Die solcherart klassifizierten Druckexemplare und die sekundären Metadaten schickt der Fachreferent wieder an das Referat 5.3 zurück **(8)**, welches die beiden klassifizierten Druckexemplare an das Referat 3.4 sendet **(9)**. Die dortigen Aktionen werden in dem bereits erwähnten separaten Sequenzdiagramm beschrieben. Die vom Fachreferenten erhaltenen sekundären Metadaten werden nun von Mitarbeitern des Referats 5.3 unter Verwendung der Administrationsanwendung „*AdminTools*“ in das System eingegeben und validiert **(10)**. Alle weiteren speziell das Referat 5.3 betreffenden Aktionen werden im zweiten Teil des Sequenzdiagramms beschrieben, welches in **Abbildung 6** zu sehen ist:

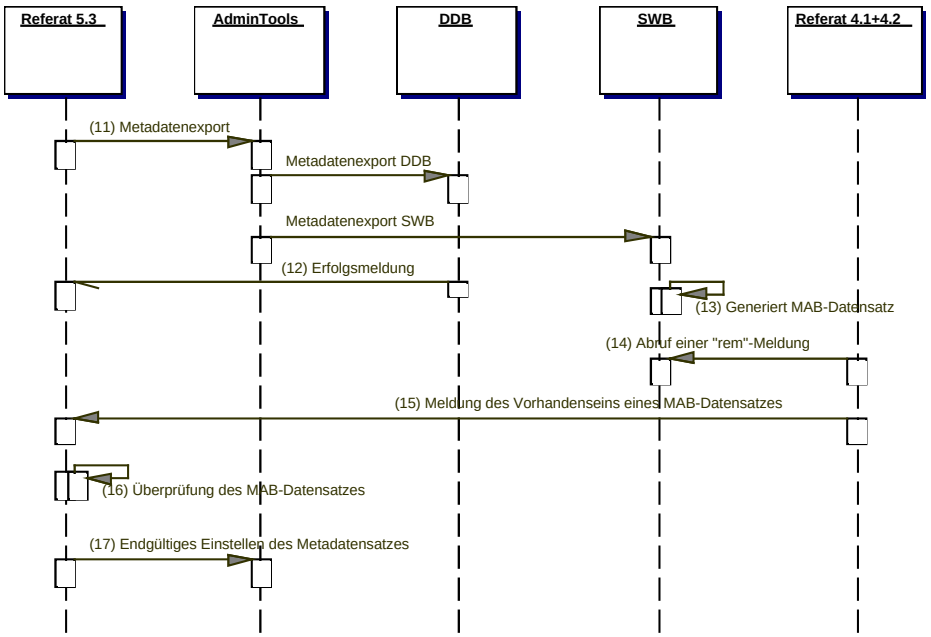

***Abbildung 6** - Sequenzdiagramm: Geschäftsgang in Bezug auf das Referat 5.3 - Teil 2*

[6]Zu den sekundären Metadaten, die der Fachreferent erstellt, gehört die Einordnung in verschiedene Klassifikationen (z. Z. DNB, RVK) sowie die Schlagwortvergabe nach Schlagwortnormdatei (SWD) (siehe auch Abschnitt A.2 auf Seite 231 im Anhang).

Wenn sowohl die primären als auch die sekundären Metadaten korrekt validiert sind, ist der Metadatensatz im bibliothekarischen Sinne vollständig, so dass er jetzt an die entsprechenden externen Institutionen exportiert werden kann. Der Metadatenexport erfolgt technisch automatisiert an die Institutionen SWB (*Südwestdeutscher Bibliotheksverbund*) und DDB (*Die Deutsche Bibliothek*) mit Hilfe der Administrationsanwendung **(11)**, wird aber von einem Mitarbeiter des Referats 5.3 manuell initiiert. Die DDB sendet im Falle eines erfolgreichen Empfangs eine Erfolgsmeldung via E-Mail **(12)**. Der SWB verschickt keine explizite Erfolgsmeldung, sondern erstellt auf der Basis der exportierten Metadaten eine Titelaufnahme für die Dissertation im MAB-Format (Maschinelles Austauschformat für Bibliotheken)[7] **(13)**. Ein solcher Eintrag kann vom Referat 4.1 bzw. 4.2 in der SWB-Katalogisierungsdatenbank erkannt werden, weil er eine entsprechende Markierung besitzt[8] **(14)**. Wird eine solche Markierung gefunden, meldet das Referat 4.1 / 4.2 das Vorhandensein dieses MAB-Datensatzes an das Referat 5.3 **(15)**. Dort wird der MAB-Datensatz überprüft und mit den Ausgangs-Metadaten verglichen (die sogenannte „Hochkatalogisierung“) **(16)**. Konnte diese Überprüfung erfolgreich abgeschlossen werden, so wird der Metadatensatz mit Hilfe der Administrationsanwendung endgültig in die persistente Datenbasis des Systems eingestellt **(17)**. Danach ist die Dissertation über die noch zu entwickelnde Webschnittstelle für Browsing, Suche und Navigation weltweit langfristig verfügbar.

Das Referat 3.4 ist der letzte Akteur, der genauer anhand von UML-Diagrammen im Kontext des Geschäftsmodells betrachtet wird. **Abbildung 7** zeigt zunächst alle für das Referat 3.4 relevanten Anwendungsfälle:

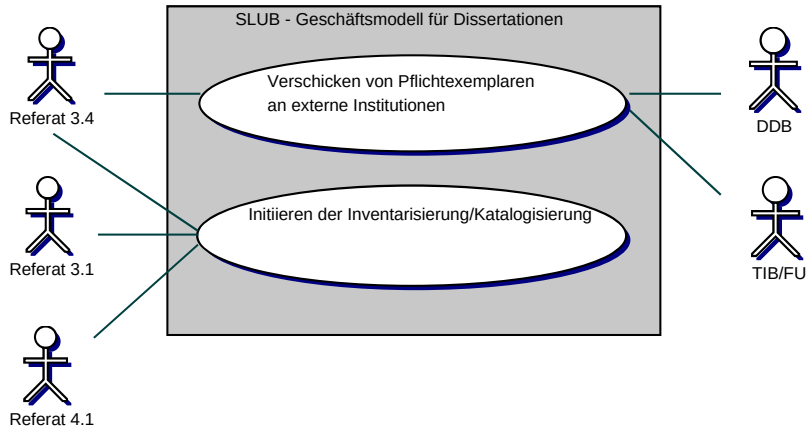

***Abbildung 7** - Anwendungsfalldiagramm: Kontext des Systems für das Referat 3.4*

[7] Außerdem erstellt er in der Verbunddatenbank auf Basis des exportierten Metadatensatzes einen neuen Eintrag.

[8] ein Abrufzeichen, die sogenannte „rem“-Eintragung

Die zwei dem Referat 3.4 zuzuordnenden Anwendungsfälle sind zum einen das Verschicken von Pflichtexemplaren (Druckexemplaren) der Dissertation an die externen Institutionen DDB (*Die Deutsche Bibliothek*) und TIB (*Technische Informationsbibliothek Hannover*) bzw. FU (*Freie Universität Berlin*) und zum anderen das Initiieren der Inventarisierung bzw. Katalogisierung der sacherschlossenen bzw. klassifizierten Druckexemplare, die im Hause verbleiben. Das in **Abbildung 8** gezeigte Sequenzdiagramm verdeutlicht die dazugehörigen Aktionen in ihrer zeitlichen Reihenfolge:

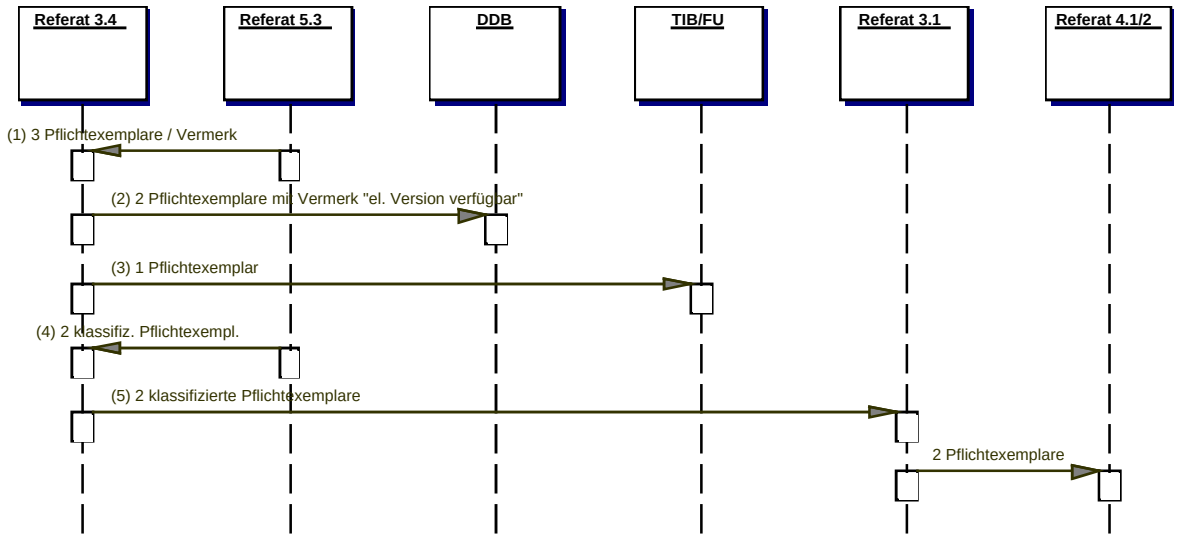

Abbildung 8 *- Sequenzdiagramm: Geschäftsgang in Bezug auf das Referat 3.4*

Das Referat 3.4 erhält zunächst vom Referat 5.3 drei Pflichtexemplare (Druckexemplare) der Dissertation mit dem Vermerk „Zu diesem Dokument ist eine elektronische Version in Kürze verfügbar“ **(1)**. Diese Exemplare werden vom Referat 3.4 an externe Institutionen verschickt. Zwei davon gehen an die DDB (*Die Deutsche Bibliothek*), zusammen mit dem genannten Vermerk **(2)**. Der Vermerk geht deshalb an die DDB, weil diese Institution ja ebenfalls die Metadaten der elektronischen Version der Dissertation erhält, allerdings (in der Regel) erst später (siehe Sequenzdiagramm Referat 5.3). Da die DDB zu dem Zeitpunkt, wenn sie die Druckexemplare erhält, nicht wissen kann, ob diese Dissertation auch elektronisch publiziert wird, ist dieser Vermerk sinnvoll, insbesondere deshalb, weil er der DDB die Chance gibt, die lokale Katalogisierung nicht anhand der spärlichen zur Druckversion verfügbaren Informationen zu unternehmen, sondern anhand der wesentlich umfangreicheren Metadaten der elektronischen Version. Ein Pflichtexemplar wird außerdem an die TIB (*Technische Informationsbibliothek Hannover*) bzw. an die FU (*Freie Universität Berlin*) verschickt[9] **(3)**. Im weiteren

[9]In Abhängigkeit vom fachlichen Gegenstand der Dissertation

Verlauf des Geschäftsganges erhält das Referat 3.4 vom Referat 5.3 zwei Pflichtexemplare, die bereits vom entsprechenden Fachreferenten klassifiziert wurden **(4)**. Für diese Exemplare initiiert das Referat die lokale Inventarisierung und Katalogisierung, indem sie zunächst an das Referat 3.1 (Inventarisierung) verschickt werden **(5)**. Von dort aus werden sie an das Referat 4.1 respektive Referat 4.2 weitergeleitet, wo die Aufnahme in den Katalog erfolgt.

3.1.2. Arbeitsablauf für Berichte

Bei dem Arbeitsablauf für Dissertationen konnten die Arbeitsgänge für die gedruckte und die elektronische Version des Dokuments sehr gut miteinander synchronisiert und verbunden werden. Dies war vor allem deshalb möglich, weil durch bestimmte gesetzliche Bestimmungen die Zeitpunkte der Abgabe von gedruckter und elektronischer Version sehr eng beieinanderliegen und im Idealfall sogar identisch sind. Bei der Dokumentenart der Berichte ist dieser Vorteil leider nicht gegeben. Es lässt sich keine Aussage über die Reihenfolge der Abgabe von gedruckten bzw. elektronischen Exemplaren eines Dokuments treffen, ja nicht einmal eine Aussage dahingehend, ob überhaupt eine Druckversion abgegeben wird. Aus diesem Grund lässt sich der Arbeitsgang der Behandlung gedruckter Berichte kaum in den für elektronische Berichte integrieren und soll auch nur am Rande behandelt werden, da der Fokus der Diplomarbeit ja auf der Behandlung elektronischer Dokumente liegt. Für den Spezialfall, dass auch ein Bericht zur gleichen Zeit sowohl gedruckt als auch in seiner digitalen Form im Referat 5.3 eintrifft, nähert sich das im folgenden gezeigte Geschäftsmodell für Berichte allerdings dem für Dissertationen an. Ansonsten ist das Geschäftsmodell für Berichte aufgrund der Tatsache, dass (fast) ausschließlich die elektronische Version des Dokuments vom Modell berücksichtigt wird, entsprechend weniger komplex und soll daher auch lediglich in zwei Sequenzdiagrammen erläutert werden. Das erste dieser Diagramme ist in **Abbildung 9** auf der folgenden Seite abgebildet.

Zunächst nutzt der Autor eines Berichtes ebenfalls die Anwendung „*MetadatenGenerator*“, um dem System die Metadaten zu seinem Dokument zu übermitteln **(1)**. Der MetadatenGenerator informiert wieder das Referat 5.3 über den Eingang eines neuen Metadatensatzes **(2)** und verschickt gegebenenfalls eine Bestätigung an den Autor **(3)**. Der Autor gibt daraufhin die elektronische Version des Dokuments bei der SLUB ab[10] **(4)**. Dort wird das elektronische Dokument durch ein formales Prüfen der Datei validiert **(5)**. Anschließend erhält der Autor eine Bestätigung oder eine

[10] unter Nutzung einer der bereits genannten Übermittlungsarten

Fehlermeldung **(6)**. Die Primär-Metadaten des Autors werden mit Hilfe der Administrationsanwendung „*AdminTools*“ validiert **(7)** und ein Ausdruck derselben an das Referat 3.4 verschickt **(8)**. Dieser Ausdruck dient dem Referat 3.4 als Information, dass ein elektronischer Bericht abgegeben wurde. So kann sich dieses Referat darum bemühen, auch ein Druckexemplar dieses Berichtes zu erlangen. Dieses kann später anhand der dann von der elektronischen Version im OPAC verfügbaren Informationen katalogisiert werden, was aber außerhalb des hier betrachteten Geschäftsmodells für Berichte liegt.

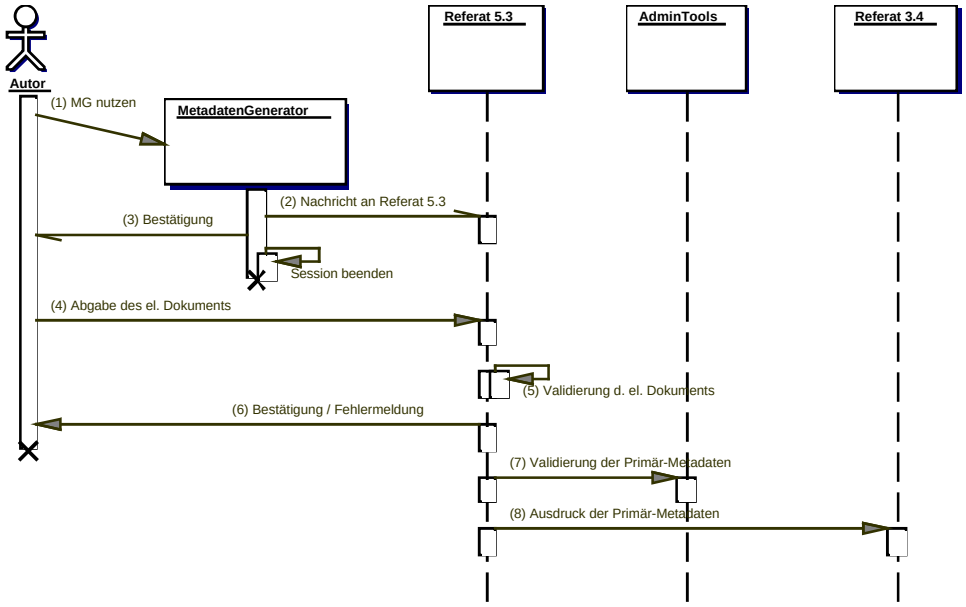

Abbildung 9 *- Sequenzdiagramm: Geschäftsgang für Berichte - Teil 1*

Abbildung 10 zeigt auf der folgenden Seite den zweiten Teil des Sequenzdiagramms. Hier wird als erstes ein Ausdruck der Primär-Metadaten an den entsprechenden Fachreferenten der Abteilung 10 bzw. 11 geschickt **(9)**. Der Fachreferent nimmt die Sacherschließung vor und liefert die entstandenen Sekundär-Metadaten an das Referat 5.3 zurück **(10)**. Diese sekundären Metadaten werden dann mit Hilfe der *AdminTools* in das System eingegeben und validiert **(11)**. Anschließend erfolgt wieder der Metadatenexport, diesmal allerdings nur an den SWB (*Südwestdeutscher Bibliotheksverbund*) **(12)**. Dort wird ein MAB-Datensatz (Maschinelles Austauschformat für Bibliotheken) erstellt **(13)**, welcher vom Referat 4.1 bzw. 4.2 durch die „rem“-Kennung als neuer Eintrag erkannt wird **(14)**. Das Referat 4.1 / 4.2 meldet daraufhin das Vorhandensein an das Referat 5.3, welches den Datensatz noch einmal überprüft („Hochkatalogisieren“) **(16)**. Als letztes wird der Metadatensatz durch die Administrationsanwendung in die persistente Datenbasisis des Systems eingestellt **(17)**. Der

Bericht ist dann weltweit und langfristig über die entsprechend noch zu realisierende Webschnittstelle verfügbar.

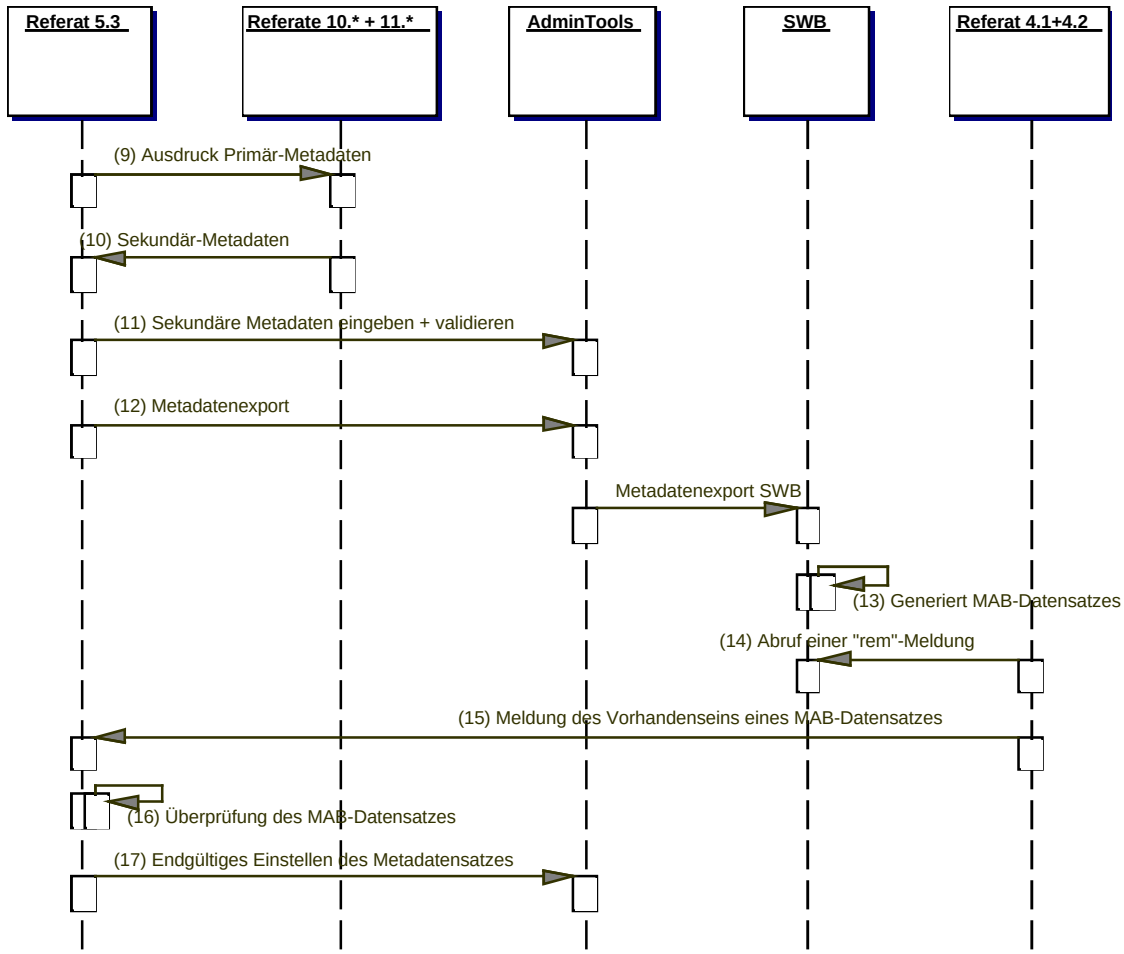

Abbildung 10 *- Sequenzdiagramm: Geschäftsgang für Berichte - Teil 2*

3.1.3. Einschätzung der Arbeitsabläufe

Im Rahmen des Diploms konnte für beide Dokumentenarten ein stimmiges und im Vergleich zu der früheren Vorgehensweise sehr geordnetes Geschäftsmodell entwickelt werden. Von fundamentaler Bedeutung war es, das für elektronische Hochschulschriften hauptsächlich zuständige Referat 5.3 in alle mit dem HSSS verbundenen Arbeitsgänge zu involvieren, so dass die komplexen Abläufe von dieser Stelle aus zentral koordiniert werden können, ohne aber die Kompetenz anderer Referate zu beschneiden. Besonders

intensiv war dies für das Geschäftsmodell der Dokumentenart Dissertation von Nöten, da hier die Zahl der beteiligten internen und externen Instanzen besonders hoch ist. Beide Geschäftsmodelle wurden mit allen beteiligten Referaten bzw. Institutionen besprochen und so weit möglich auf die jeweiligen Verhältnisse respektive Bedürfnisse abgestimmt. Zusammenfassend lässt sich sagen, dass mit den beiden hier vorgestellten Geschäftsmodellen und den damit verbundenen Arbeitsabläufen eine Systemorganisation gefunden wurde, die als Ausgangspunkt für die weiteren Entwicklungen im Rahmen der Diplomarbeit ausgezeichnet geeignet ist und auch mittelfristig für die Beschreibung aller notwendigen, den Hochschulschriftenserver der SLUB betreffenden Aktionen ausreichen sollte.

Natürlich gibt es in beiden Geschäftsmodellen noch Verbesserungspotenzial. Ein erster Kritikpunkt betrifft die Übermittlung der Primär-Metadaten an den entsprechenden Fachreferenten. Hier wurde auf Wunsch der SLUB eine Lösung vorgesehen, die das Ausdrucken der bereits im System befindlichen Primär-Metadaten und das anschließende Verschicken über die Haus-Post beinhaltet. Die Übermittlung der sekundären Metadaten vom Fachreferenten zurück zum Referat 5.3 folgt dem selben Prinzip. Das ist natürlich sehr ineffektiv, wenn man bedenkt, dass an zwei Stellen bereits digitalisierte Daten in einem manuellen Vorgang wieder vom Papier zurück in das System gebracht werden. Ein erster Kompromiss, der an dieser Stelle vorgesehen wurde, ist das Versenden der primären bzw. sekundären Metadaten als E-Mail, was von der Administrationsanwendung „*AdminTools*“ gewährleistet werden müsste. In diesem Fall könnte der entsprechende Mitarbeiter des Referats 5.3 die Sekundär-Metadaten wenigstens über „kopieren“ und „einfügen“ in das System[11] einbringen. Noch besser[12] wäre die Realisierung einer direkten Interaktionsmöglichkeit für die Fachreferenten in Bezug auf die Administrationsanwendung. Dies würde bedeuten, dass auch sie Zugang zu dieser Schnittstelle hätten und somit die entsprechenden Aktionen (Erstellung der sekundären Metadaten) direkt durchführen könnten. Technisch steht einer solchen Lösung nichts im Wege[13], allerdings erfordert sie einen gewissen organisatorischen Aufwand, der sich unter anderem auch durch notwendige Schulungen bzw. Weiterbildungen ergibt.

Ein weiterer Kritikpunkt, zudem informationstechnisch von viel größerer Tragweite, ist die Tatsache, das Druckexemplar und elektronische Version desselben Dokuments aus bibliothekarischem Blickwinkel als unterschiedliche Ressourcen betrachtet werden und damit auch zweimal (und sehr wahrscheinlich unterschiedlich) katalogisiert wer-

[11] in diesem Fall über die AdminTools-Schnittstelle

[12] und möglicherweise in einer weiteren Ausbaustufe des HSSS realisierbar

[13] im Gegenteil, sie ist sogar relativ leicht realisierbar, da die gesamte Architektur auf ein verteiltes Arbeiten ausgelegt ist

den. Hier müsste strikt (falls überhaupt notwendig) zwischen der *Repräsentation* und dem *Inhalt* einer Ressource unterschieden werden. Der Inhalt der elektronischen Version und des Druckexemplars einer Ressource ist nämlich derselbe (zumindest sollte es so sein), daher sollten natürlich auch alle primären und sekundären Metadaten *identisch* sein, so dass sich eine zweimalige Katalogisierung eigentlich von selbst verbietet. Die Repräsentation einer Ressource kann sich hingegen sehr wohl unterscheiden und erfordert gegebenenfalls auch unterschiedliche Beschreibungen, dann allerdings getrennt vom Inhalt oder jeweils mit dem Verweis auf *dieselbe* Inhaltsbeschreibung. Leider ist es aber gängige Praxis, für jede Repräsentation einer Ressource bzw. im Printbereich sogar für jede Auflage einer Ressource eine komplett neue Titelaufnahme zu erstellen, wie **Abbildung 11** zeigt:

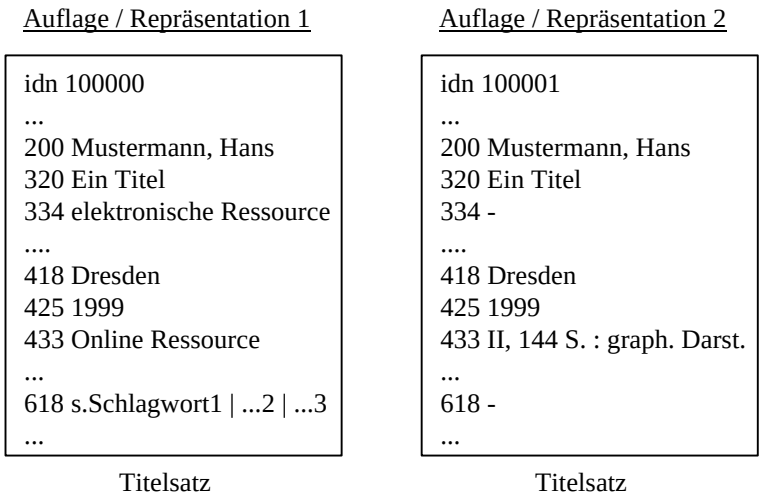

***Abbildung 11** - Doppelte Titelaufnahme einer Ressource*

Diese Abbildung zeigt Ausschnitte des MAB-Datensatzes zweier Repräsentationen (elektronische Version, Druckexemplar) ein und derselben Ressource. Wie ersichtlich ist, liegen große Teile (z. Bsp. die MAB-Felder 200, 320, 418 ...) des Datensatzes redundant vor. Dies birgt potentiell die Gefahr von Inkonsistenzen und nichtdeterministischen Suchergebnissen, falls diese Felder - anders als hier gezeigt - eben nicht einhundertprozentig identisch sind. Andere Felder sind entweder in dem einen oder in dem anderen Datensatz enthalten (Felder 334, 618). Bis hierhin wäre der eine (zweite) Datensatz also eine echte Teilmenge des anderen (ersten). Das Feld 433 kommt in beiden Datensätzen mit jeweils unterschiedlicher Belegung vor. Falls dieses Feld mehrfach belegt werden dürfte, könnten die beiden Datensätze wie in **Abbildung 12** gezeigt zusammengefasst werden. Dies hätte für das Druckexemplar, sofern es noch nicht sachlich erschlossen wurde, den großen Vorteil, dass jetzt auch für diese Repräsentation der Ressource erweiterte Infomationen wie Schlagworte verfügbar sind.

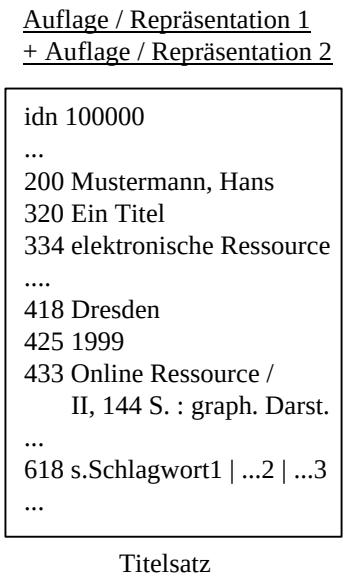

Abbildung 12 *- Einheitliche Titelaufnahme einer Ressource*

Falls eine direkte Zusammenfassung technisch nicht möglich ist, muss strikt zwischen der Inhaltsbeschreibung der Ressource (für alle Repräsentationen gleich) und der Repräsentationsbeschreibung (für jede Repräsentation unterschiedlich) unterschieden werden. **Abbildung 13** zeigt eine Lösungsmöglichkeit für diesen Fall:

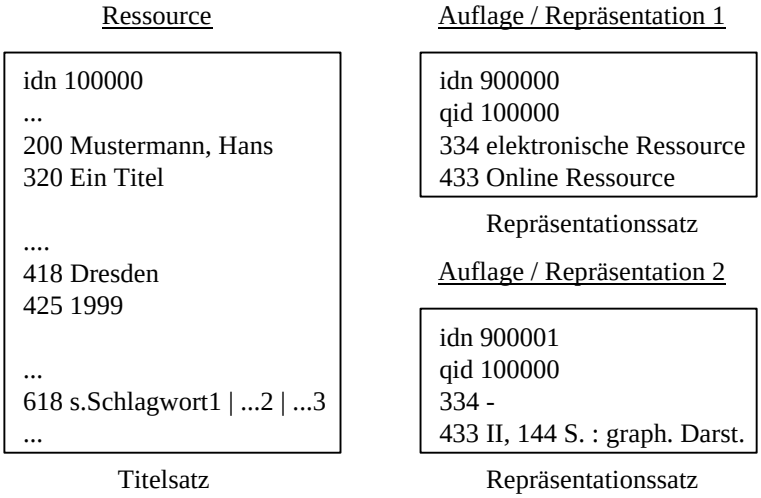

Abbildung 13 *- Trennung von Inhalt / Repräsentation*

Wie die Abbildung zeigt, existiert zu jeder Ressource nur *eine* Beschreibung des Inhalts (hier der MAB-Datensatz mit der Identifikationsnummer 1000000). Für die

unterschiedlichen Merkmale der Repräsentationen müssen separate Datensätze mit eigenen Identifikationsnummern erstellt werden, die aber alle auf *dieselbe* Inhaltsbeschreibung verweisen (über „qid“ auf den MAB-Datensatz mit der Identifikationsnummer 100000).

Informationstechnisch gesehen wäre eine einheitliche Quellenbeschreibung (egal ob mit oder ohne Auslagerung der Repräsentationsinformationen) einer Ressource ein großer Fortschritt. In der Praxis sind dazu mit Sicherheit einige Hindernisse zu überwinden. Angesichts der zunehmenden Verlagerung der Publizistik in den elektronischen Bereich und den zunehmend zu erwartenden Dopplungen sollte eine solche Umstellung jedoch nicht in allzu ferne Zukunft verschoben werden, erste Bemühungen in diese Richtung sind zum Teil auch schon unternommen worden (so heißt es beispielsweise in [SWB00] , dass bei parallelen Ausgaben in der Titelaufnahme der konventunellen Ausgabe einen Link auf die Online verfügbare Version zu setzen, was allerdings nicht ausreicht). Zum Aufwand einer solchen Umstellung kann an dieser Stelle keine Aussage getroffen werden, da der Autor mit dem entsprechenden bibliothekarischen Einzelheiten zu wenig vertraut ist. Fakt ist aber, dass die gängige Praxis, für jede Repräsentation oder Auflage[14] einer Ressource eine neue Titelaufnahme zu erstellen, informationstechnisch betrachtet schlicht und einfach *falsch* ist.

[14] Die Aussage trifft nur für unveränderte Auflagen zu, „geänderte“ Auflagen machen gegebenenfalls auch eine erneute Titelaufnahme notwendig.

3.2. Architekturübersicht

In diesem Abschnitt soll die Gesamtarchitektur des HSSS auf einem relativ hohem Abstraktionsniveau beschrieben werden, um das System überblicksartig vorzustellen. Dies geschieht deshalb noch vor der Erläuterung der Architekturvisionen, weil sich diese nicht auf das grobe Architekturschema oder das Zusammenspiel der einzelnen Komponenten[15] beziehen, sondern auf Gegenstände eines höheren Detaillierungsgrades. **Abbildung 14** zeigt das Komponentenmodell als eine der größtmöglich sinnvollen Architekturabstraktion:

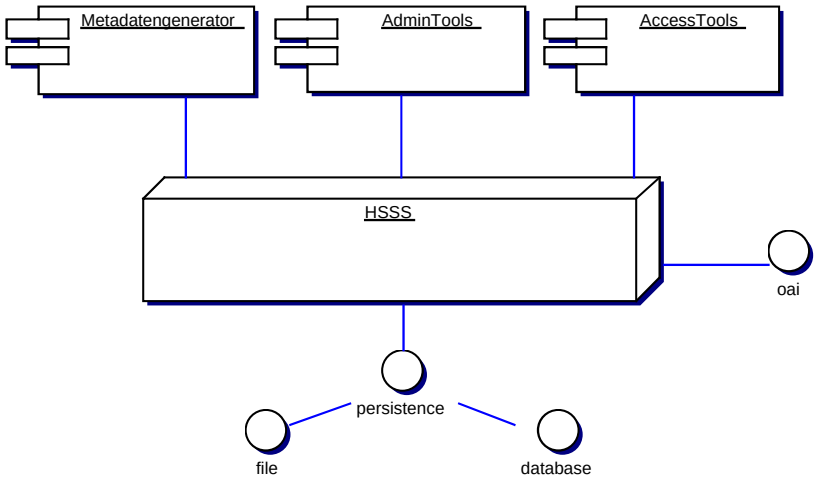

Abbildung 14 *- Komponentenmodell der HSSS-Gesamtanwendung*

Das Gesamtsystem des HSSS besteht demnach im Wesentlichen aus den drei Komponenten *„MetadatenGenerator“*, *„AdminTools“* und *“AccessTools“*. Bei dem *„MetadatenGenerator“* handelt es sich um die bereits in der Belegarbeit zu großen Teilen realisierte Anwendung, die es Autoren der TU Dresden auf eine bequeme Art und Weise erlaubt, die Metadaten zu ihrer Hochschulschrift zu erstellen und an die SLUB zu übermitteln. Die Systemkomponente *„AdminTools“* erlaubt den Mitarbeitern des Referats 5.3 den gesamten Arbeitsablauf softwaregestützt zu begleiten und zu koordinieren. Vor allem die Aktionen, die die Bearbeitung der Metadaten betreffen (Validieren der Primär-Metadaten, Erstellen und Validieren der Sekundär-Metadaten, Exportieren der Metadaten und Einstellen in die persistente Datenbasis des Systems),

[15]Die Architektur des HSSS auf dem Abstraktionsniveau des Zusammenspiels der einzelnen Komponenten oder der Technologieplattform ist bereits im Großen Beleg ausführlich definiert worden[Schb00] und wird sich für die Diplomarbeit nicht ändern.

lassen sich über diese Schnittstelle realisieren. Mit dem Arbeitstitel *„AccessTools“* wird die in der Diplomarbeit zu realisierende webbasierte Schnittstelle zur Suche und Navigation über den Inhalten des Hochschulschriftenservers bezeichnet. Das Gesamtsystem besitzt außerdem eine lose gekoppelte Persistenzschnittstelle. Im Beleg war die Persistenzschicht dateibasiert, im Rahmen des Diploms soll diese Lösung beibehalten, jedoch zusätzlich um eine datenbankbasierte Persistenzlösung erweitert werden. Außerdem existiert eine OAI-Schnittstelle, welche das *OAI Metadata Harvesting Protocol* implementiert und damit die Grundlage schafft, dass das Gesamtsystem in einem Verbund von OAI-Archiven als *Data Provider* auftreten kann.

Abbildung 15 zeigt das Gesamtsystem auf einem ähnlich hohem Abstraktionsniveau wie die vorherige Abbildung, aber mit einem anderen Fokus. Während die vorige Abbildung sich auf die selbst zu entwickelnden Komponenten des Systems konzentrierte, soll die folgende Abbildung das Zusammenspiel mit externen Komponenten, API's und Services in einem Schichtenmodell verdeutlichen:

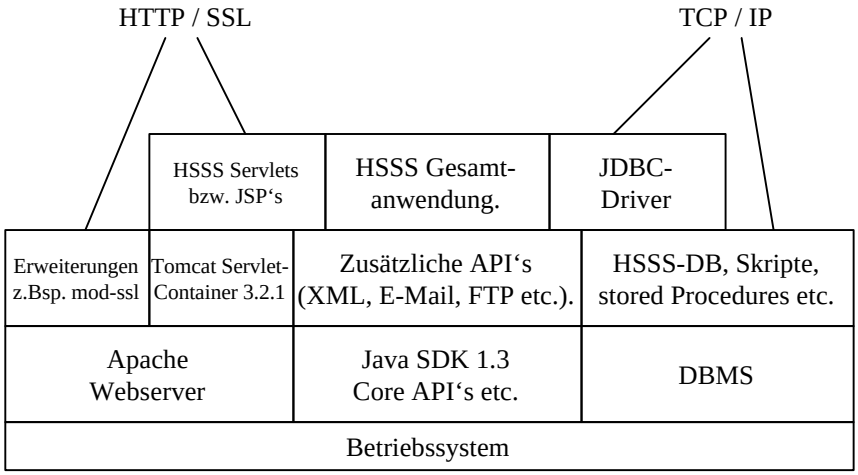

Abbildung 15 *- Systemkomponenten im Schichtenmodell*

Auf der Betriebssystemschicht setzen zum einen der Apache Webserver, die Java Entwicklungsumgebung inklusive aller Standard-API's sowie das Datenbankverwaltungssystem auf. Diese drei Systemkomponenten bilden die wesentlichen Bestandteile des HSSS-Gesamtsystems und sind als Grundlage aller weiteren Komponenten unverzichtbar. Auf dem Datenbankverwaltungssystem setzt die eigentliche Datenbank des HSSS auf, die im Rahmen des Diploms noch zu realisieren ist. Die Java-Entwicklungsumgebung mit ihren Standard-API's wird durch eine Reihe weiterer API's erweitert werden müssen, um die geforderten Dienste erbringen zu können. Der *Apache*-Webserver wird ebenfalls durch einige Erweiterungen ergänzt werden, zum

Beispiel durch ein Modul (*„mod-ssl“*), welches ihn SSL-fähig macht. Entscheidender aber ist der Servlet-Container *„Tomcat“*[L-TC] des *„Jakarta“*-Projektes[L-JA] der *„Apache Software Foundation“* (ASF) [L-AP], der erst die Server-seitige Ausführung der Anwendungen ermöglicht. Das Zusammenspiel von *Apache* und *Tomcat* wurde in [Schb00] ausführlich beschrieben. Auf dem Servlet-Container setzen die für den HSSS entwickelten Servlets bzw. Java ServerPages (JSP's) auf, welche ebenfalls mit der in Java realisierten HSSS-Gesamtanwendung interagieren, welche wiederum über einen entsprechenden JDBC-Datenbanktreiber auf die Datenbank zugreifen kann.

Die folgenden Abschnitte dieses Kapitels beschreiben die Architektur des HSSS etwas mehr im Detail, wobei der Fokus selbstverständlich auf den eigenen Entwicklungen im Bereich der Java-basierten Gesamtanwendung, der Servlets respektive Java ServerPages bzw. des Datenbankschemas liegt. Zunächst werden jedoch Architekturvisionen vorgestellt, welche die Vorstellung von einem idealen Gesamtsystem vermitteln sollen und dadurch mögliche Schwerpunkte für den anschließenden Entwurfsprozess aufzeigen.

3.2.1. Architekturvision

Die hier vorgestellten Architekturvisionen betrachten die Gesamtanwendung und vor allem deren Entstehungsprozess und diverse Architekturdetails vorerst völlig losgelöst von praktischen Gesichtspunkten wie etwa der Realisierbarkeit. Da es sich um „Visionen“ handelt, ist das auch nicht notwendig. Erst der sich anschließende Abschnitt wird die entstandenen Visionen auf ihre Praxistauglichkeit und Realisierbarkeit, ihren Nutzen und ihre Kosten hin überprüfen und gegebenenfalls einschränken. Die Aussagen des ersten Teils dieses Abschnitts beziehen sich auf die Anwendungen *MetadatenGenerator* und *AdminTools*, da diese auch in einem engen thematischen Zusammenhang stehen. Der zweite Teil wird sich mit den *AccessTools* beschäftigen.

Integraler Bestandteil und Bezugspunkt der ersten beiden Anwendungen ist die Beschreibung eines Dokuments durch einen Metadatensatz in einem bestimmten Metadatenformat. Für den HSSS wird ein Metadatenformat nach dem *Dublin Core Element Set* verwendet [L-DC]. Ein Metadatensatz in diesem Format besteht derzeit aus 15 Elementen (Kategorien), die jeweils durch Unterelemente (Subelements) verfeinert werden können. Dadurch entsteht eine streng hierarchische Struktur. Eine Forderung an den HSSS war, dass er möglichst XML-fähig sein sollte, zumindest aber sollte aufgezeigt werden, an welchen Stellen der Einsatz dieser Technologie besonders sinnvoll

ist. Bei der Repräsentation von Metadatensätzen ist ein solcher Einsatz zum ersten Mal zu überdenken. Zur Laufzeit wird ein Metadatensatz natürlich durch eine Objekt-Struktur im Hauptspeicher gebildet, wie aber wird er temporär zwischengespeichert? In der Belegarbeit wurde die temporäre Persistenz durch Bord-Mittel von Java in Form einer Objektserialisierung[16] respektive -deserialisierung realisiert. Das hat einige Vorteile[17], aber auch gravierende Nachteile: so ist die Serialisierung stark von der Struktur des Objektes abhängig. Wenn sich diese zu stark ändert, können frühere Serialisierungen unter Umständen nicht wieder korrekt deserialisiert werden, was im schlimmsten Fall einen Datenverlust bedeuten könnte. Außerdem ist eine Serialisierung von der verwendeten Programmiersprache (in diesem Fall Java) und sogar deren Version abhängig und somit nicht universell lesbar oder austauschbar. Die Vorteile von XML an dieser Stelle bestehen also insbesondere darin, dass sich ein solcherart repräsentierter Metadatensatz universell austauschen und im Notfall auch extern bearbeiten ließe und sogar bei einem kompletten Wechsel des Metadatenformats relativ leicht angepasst werden könnte. Um diese Vorteile zu realisieren, müssten die Metadatensätze des HSSS also bei der temporären Speicherung von einer Objektstruktur in eine XML-Struktur abgebildet werden (bzw. umgekehrt), wie **Abbildung 16** verdeutlicht:

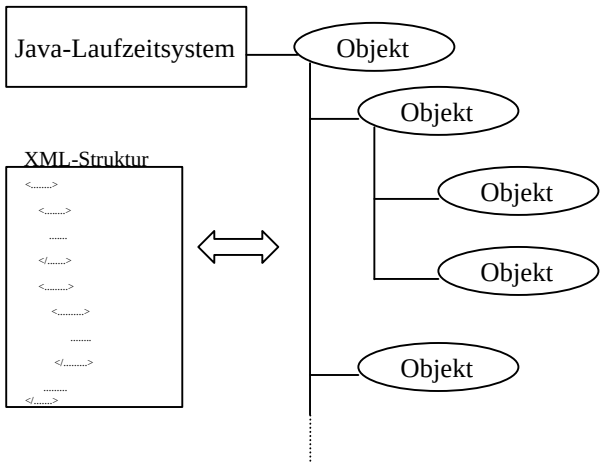

Abbildung 16 *- Abbildung Objektstruktur - XML-Struktur*

Der *MetadatenGenerator* würde in diesem Szenario also am Ende der Interaktionen mit dem Autor die Repräsentation des Metadatensatzes von einer Objektstruktur im Hauptspeicher in eine (temporär) persistente XML-Struktur umwandeln, die sich in einer Datei oder in einer Datenbank befindet. Aus dieser Datenquelle würden die *AdminTools* die XML-Struktur mit Hilfe eines geeigneten Parsers wieder einlesen und in

[16] Objektserialisierung bedeutet, dass ein im Hauptspeicher befindliches Objekt inklusive aller seiner nicht-transienten Attribute und Methoden in Form einer binären Datei gespeichert wird.

[17] Man benötigt keine zusätzlichen Technologien und der Vorgang ist relativ schnell.

eine adäquate Java-Objektstruktur im Hauptspeicher zurückwandeln. Bei jeder weiteren (i.d.R. inkrementellen) Änderung des Metadatensatzes in den *AdminTools* würde sich dieser Vorgang wiederholen. Die Repräsentation als XML-Struktur ist dabei aufgrund der hierarchischen Struktur des *Dublin Core Element Set* relativ problemlos und sollte formal etwa wie in **Listing 4** aussehen:

```
1  <?xml version="1.0" encoding="ISO−8859−1" ?>
2  <metadataset>
3      <dc−element>
4          .....
5      </dc−element>
6      <dc−element>
7          <dc−subelement>
8              .....
9          </dc−subelement>
10         ...
11     </dc−element>
12     ...
13     <temp>
14         <temp−element>
15             ....
16         </temp−element>
17     </temp>
18 </metadataset>
```

***Listing 4** : formale XML-Struktur eines Metadatensatzes*

Wie dem Listing zu entnehmen ist, beinhaltet ein Metadatensatz zu diesem Zeitpunkt nicht nur die Metadaten nach dem *Dublin Core*-Format, sondern auch temporäre Metadaten, für die es keine *Dublin Core*-Entsprechung gibt[18]. Die erste Architekturvision lautet also:

Architekturvision 1: *Bis zum endgültigen Einstellen von Metadatensätzen erfolgen alle temporär persistenten Manifestationen der Metadatensätze als XML-Struktur, unabhängig von der verwendeten Speicherungsart (dateibasiert, datenbankbasiert, etc.).*

Die sich daraus ergebenden Vorteile sind insbesondere:

- Plattformunabhängiges sowie sprachunabhängiges, flexibles, über Systemgrenzen hinweg austauschbares Persistenzformat für temporäre Metadaten
- Entkopplung Objektstruktur - Persistenzstruktur, insbesondere ist die Objektstruktur beliebig änderbar, weil sich die zugehörige XML-Abbildung relativ einfach ändern lässt

[18]Dabei handelt es sich hauptsächlich um technisch notwendige temporäre Metadaten, die zum Beispiel Stati innerhalb der Arbeitsabläufe beschreiben etc.. Diese Daten werden nicht endgültig persistent eingestellt.

- Temporäre Daten sind auch außerhalb der vorgesehenen Anwendung verwertbar und verarbeitbar, im Notfall sogar direkt manuell
- Schon zu diesem Zeitpunkt lassen sich aus dem XML-basierten Ausgangsformat der Metadaten relativ leicht diverse Zielformate (z. Bsp. XML, PDF, HTML usw.) generieren, etwa für Reports etc.

Alle diese Vorteile führen bei genauerer Betrachtung vor allem zu einer wesentlich erhöhten Robustheit und Wiederverwendbarkeit der Gesamtanwendung und insbesondere der temporär persistenten Metadaten. Als kleiner Nachteil ergibt sich lediglich der Aufwand (zeitlich, programmiertechnisch) für das Parsen der XML-Struktur, der allerdings in einem erträglichen Rahmen liegt.

Die zweite Architekturvision entwickelt sich ebenfalls aus der Betrachtung der ersten zwei Anwendungen in Verbindung mit dem Metadatensatzformat. Letztendlich bestimmen Art, Umfang und Struktur des Metadatensatzes große Teile dieser Anwendungen, sowohl was deren interne Struktur, aber auch externe Merkmale wie Layout der Benutzerschnittstelle und Flusskontrolle etc. betrifft. In **Abbildung 17** wird dies beispielhaft verdeutlicht, vor allem durch die Beziehung Metadatensatzbeschreibung - grafische Benutzerschnittstelle - Server-seitige Anwendung:

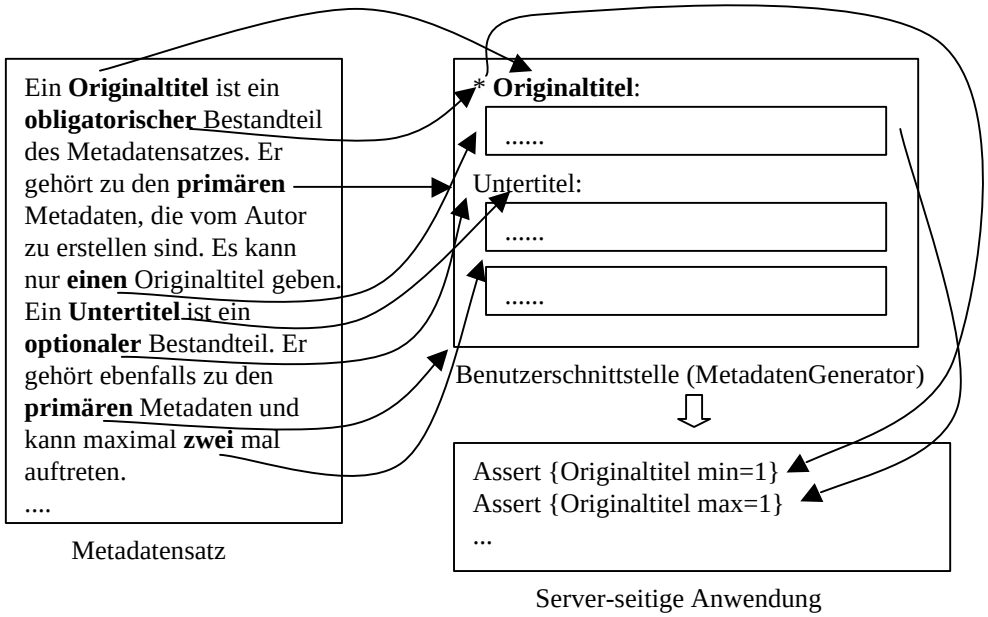
Ein **Originaltitel** ist ein **obligatorischer** Bestandteil des Metadatensatzes. Er gehört zu den **primären** Metadaten, die vom Autor zu erstellen sind. Es kann nur **einen** Originaltitel geben. Ein **Untertitel** ist ein **optionaler** Bestandteil. Er gehört ebenfalls zu den **primären** Metadaten und kann maximal **zwei** mal auftreten.
....
Metadatensatz
* **Originaltitel**:
.....
Untertitel:
.....
.....
Benutzerschnittstelle (MetadatenGenerator)
Assert {Originaltitel min=1}
Assert {Originaltitel max=1}
...
Server-seitige Anwendung

***Abbildung 17** - Einfluss der Metadatensatzbeschreibung auf die Anwendung*

Wie aus der Abbildung ersichtlich ist, hat die Metadatensatzbeschreibung einen erheblichen Einfluss auf die grafische Benutzerschnittstelle, aber auch auf die eigentliche Anwendung. So beeinflusst beispielsweise die Festlegung, ob ein Element bzw. Unterelement des Metadatensatzes optional oder obligatorisch ist, sowohl dessen Präsentation in der grafischen Benutzerschnittstelle[19] als auch dessen Behandlung (Validierung) durch die Server-seitige Anwendung[20]. Neben den in der Abbildung gezeigten lassen sich leicht weitere Bestandteile einer Metadatensatz-Spezifikation ausmachen, welche die gesamte weitere Anwendung sinnvoll beeinflussen könnten. **Tabelle 3** zeigt solche Bestandteile einer Metadatensatz-Spezifikation und wie sie das Verhalten und die Struktur der Benutzerschnittstelle/Anwendung beeinflussen könnten.

Spezifikationsbestandteil	Benutzerschnittstelle	Anwendung
Metadatensatzstruktur allgemein	Einfluss auf Gesamtstruktur der Benutzerschnittstelle	Einfluss auf die Objektstruktur
Element-Attribute wie Beschriftung, Kurzerklärung etc.	Auswirkung auf das Layout, z. Bsp. Beschriftung der Felder etc.	Kein Einfluss
Element-Attribute wie obligatorisch/optional, maximale Anzahl, maximale Länge etc.	Einfluss auf die Art und Weise der Darstellung	Bestimmt, wie die Metadaten zu validieren sind
Element-Attribute wie primary, secondary oder system	Entscheiden, ob Elemente überhaupt präsentiert werden	Entscheiden über zeitliche Positionierung / Behandlung / Bearbeitbarkeit innerhalb des Arbeitsablaufs
Element-Attribute, die die Kommunikation zwischen Benutzerschnittstelle und Anwendung beschreiben	Beeinflusst Benennung von Parametern	Erlaubt die Selektion von „Metadaten“-Parametern

***Tabelle 3** - Einflussmöglichkeiten der Metadatensatzbestandteile*

Selbstverständlich muss der Metadatensatz in einer geeigneten Art und Weise strukturiert bzw. beschrieben sein, um der Anwendung die genannten Informationen zur Verfügung stellen zu können. Hier eignet sich wieder XML ausgezeichnet, um den Metadatensatz in einer der Anwendung bekannten Struktur zu definieren und anschließend von dieser eingelesen und in eine interne Repräsentation umgewandelt zu werden. **Listing 5** zeigt eine mögliche Struktur einer solcherart in XML formulierten Metadatensatzbeschreibung.

[19] z. Bsp. ein Stern für obligatorische Elemente oder eine besondere Farbe

[20] Für obligatorische Elemente muss z. Bsp. von der Anwendung die Einhaltung der Untergrenze 1 garantiert werden.

```
1  <?xml version="1.0" encoding="ISO−8859−1" ?>
2  <metadataset documenttype="...">
3      <dc−element>
4          <name>....</name>
5          <param>....</param
6           ....
7          <dc−subelement>
8              <name>....</name>
9              <param>....</param>
10              ....
11         </dc−subelement>
12          ....
13     </dc−element
14      ....
15     <temp>
16         <temp−element>
17             <name>....</name>
18             <param>....</param
19              ....
20         </temp−element>
21          ....
22     </temp>
23 </metadataset>
```

***Listing 5** : formale XML-Struktur einer Metadatensatzbeschreibung*

Eine solche XML-Struktur müsste für jede Dokumentenart des Systems existieren, da für jede Dokumentenart unter Umständen ein separater Metadatensatz spezifiziert ist. Die Anwendung würde sie zu gegebener Zeit einlesen, parsen und die Informationen in entsprechenden Objekten ablegen. Wie gezeigt, können im Prinzip die gesamten Validatorobjekte und Großteile der grafischen Benutzerschnittstelle automatisch aus diesen Informationen erstellt werden.

In einer Idealvorstellung lassen sich sogar *alle* Metadatensatz-abhängigen Teile der ersten beiden Anwendungen komplett automatisiert und zur Laufzeit erstellen. In diesem Fall müsste die XML-Struktur natürlich zusätzliche (sehr komplexe) Informationen enthalten, die es u.a. ermöglichen, alle mit Metadaten in Zusammenhang stehenden Klassen zur Laufzeit zu bilden, zu kompilieren, auszuführen und zu speichern. Die Anwendung bestünde in diesem Fall aus einem relativ kleinen, unabhängigen statischen Teil, welcher in Abhängigkeit der zu unterstützenden Dokumentenarten (bzw. der entsprechenden Metadatensatzbeschreibungen) den weitaus umfangreicheren dynamischen Teil zu Laufzeit erstellen würde. Dies gilt auch für die automatische Erstellung von Speicherstrukturen für die endgültige Persistenzrealisierung[21]. Im Idealfall könnte man ohne tiefere System- oder Programmierkenntnisse mit Hilfe einer weiteren

[21] Zum Beispiel die automatisierte Generierung von DML/DDL-Skripten zur Erzeugung eines adäquaten Datenbankmodels/Datenbankschemas für den entsprechenden Metadatensatz respektive die durch ihn beschriebene Dokumentenart.

Schnittstelle eine beliebige Metadatensatz-Spezifikation erstellen, aus der (transparent für den Nutzer) automatisch eine entsprechende XML-Struktur erzeugt wird und anschließend alle zur Realisierung notwendigen Teile der Anwendung, ohne dass eine Zeile Quellcode programmiert werden müsste. Die zweite Architekturvision lautet daher:

Architekturvision 2: *Alle Informationen, die einen Metadatensatz beschreiben, sind von der Anwendung zu entkoppeln und in einer geeigneten, XML-basierten Struktur anzuordnen. Mit Hilfe dieser Struktur sollen möglichst alle dynamischen Teile der Anwendung automatisiert zur Laufzeit erstellt und koordiniert werden.*

Ziel dieser Architekturvision ist es, die Anwendung so flexibel wie möglich zu halten, insbesondere was die Änderung und Erweiterung von Systemeigenschaften betrifft, ohne manuell Quellcode verändern und kompilieren zu müssen. Diese Vision setzt voraus, dass auch an anderen Stellen des Gesamtsystems Informationen so weit wie möglich entkoppelt und damit konfigurierbar gehalten werden.

Die dritte Architekturvision betrifft auch den dritten Teil der Gesamtanwendung, der die Suche und die Navigation über den persistent eingestellten Metadatensätzen ermöglichen soll. Diese webbasierte Anwendung wird potentiell im Vergleich zu den anderen beiden von noch wesentlich mehr Nutzern besucht werden, so dass Überlegungen zu einem nutzerfreundlichen Layout dieser Anwendung sehr wichtig sind. Wie die Entwicklung des *MetadatenGenerator* im Rahmen der Belegarbeit gezeigt hat, bestehen die größten Design-Probleme bei Web-Anwendungen in der notwendigen Unterstützung unterschiedlicher Browser/Browserversionen und den unterschiedlichen lokalen Systemgegebenheiten der Nutzer, über die streng genommen keine Annahmen getroffen werden dürfen. Insbesondere die Bildschirmauflösung ist eine zur Zeit sehr stark variierende Größe[22], die einen nicht zu unterschätzenden Einfluss auf die Nutzbarkeit einer Web-Seite hat und deshalb beim Design unbedingt zu beachten ist. Eine Lösungsmöglichkeit besteht in der Wahl eines festen, absoluten Designs etwa für 800*600 Bildpunkte[23]. Damit sind zwar die meisten Nutzer in der Lage, das Angebot einigermaßen benutzerfreundlich (ohne horizontales „scrollen“) in ihren Browsern zu betrachten, allerdings verschwendet das Design unter Umständen 75% des vorhandenen Platzes bei Nutzern mit einer 1600*1200-Auflösung. Die Wahl eines flexiblen Layouts mit prozentualen Werten dagegen erlaubt nur eine bedingte, nicht determinierte Kontrolle des Layouts und ist ebenfalls nicht ideal. Die Verwendung einer Mischung von absoluten und prozentualen Angaben, insbesondere der zusätzliche Einsatz von DHTML[24] ist

[22]Sie schwankt von 640*480 bis 1600*1200 Bildpunkten, also um den Faktor 2,5

[23]So geschehen für den *MetadatenGenerator*

[24]Bei DHTML (Dynamic HTML) handelt es sich um einen Sammelbegriff für verschiedene Lösungen,

äußerst fehleranfällig und wartungsintensiv. Die beste Lösung besteht nach Meinung des Autors darin, Server-seitig eine beliebige Skalierung der Benutzerschnittstelle zu ermöglichen, wobei das Layout so konsistent wie möglich bleiben sollte. Die Wahl der Auflösung sollte dabei nicht von der Anwendung anhand der tatsächlichen oder vermuteten Bildschirmauflösung getroffen werden[25], sondern vom Nutzer bestimmt werden können. Die dritte, sich auf die mit *AccessTools* betitelte Teilanwendung beziehende Architekturversion lautet daher:

Architekturvision 3: *Der Nutzer der dritten Web-Anwendung kann diese in ihrer Auflösung flexibel ändern, wobei das verwendete Layout in sich konsistent bleibt.*
Dies setzt voraus, dass alle Inhalte (auch Schrift, Bilder etc.) Server-seitig skaliert werden.

Diese dritte Teilanwendung realisiert dabei die Suche (Volltextsuche, Feldsuche) und die Navigation über den Metadaten. Sie sollte dazu insbesondere für die Suche eine übersichtliche und leicht verständliche Schnittstelle besitzen, welche eine von Suchmaschinen gewohnte Funktionalität bietet[26]. Da auf der Serverseite eine moderne Architektur verwendet wird, welche die Behandlung von HTTP-Sitzungen relativ einfach ermöglicht und zudem im Laufe der Diplomarbeit eine Datenbank für das Gesamtsystem zur Verfügung steht, besteht die Möglichkeit, die dritte Teilanwendung als "Portal-Lösung" zu konzipieren[27], womit dabei insbesondere der Personalisierungsaspekt gemeint ist. Dazu ist die Möglichkeit zu schaffen, dass sich beliebige Nutzer bei dem System dauerhaft registrieren können. Für diese Nutzer können dann Informationen gespeichert werden, die ihnen eine personalisierte Sicht[28] auf die Anwendung erlauben. Dadurch lässt sich das Angebot besonders nutzerfreundlich und vor allem individuell gestalten, was die Akzeptanz, aber auch die Effektivität der Anwendung deutlich erhöhen kann. Die vierte Architekturvision lautet daher:

Architekturvision 4: *Die dritte Teilanwendung soll als Portal-Lösung realisiert werden. Dazu gehört die Integration externer Inhalte, die*

um HTML-Elemente clientseitig zur Laufzeit dynamisch zu ändern. Die Ansätze sind dabei (leider) für die jeweiligen Browser verschieden. Quellen:
`http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml.asp` (Microsoft)
`http://developer.netscape.com/tech/dynhtml/` (Netscape)

[25] Was über JavaScript mit Einschränkungen realisierbar wäre

[26] Dazu zählt die Realisierung von mindestens zwei Detailstufen bei der Ergebnisdarstellung sowie die Unterstützung von Teilergebnissen (Blättern im Gesamtergebnis)

[27] So könnten andere, in Beziehung stehende Inhalte der SLUB integriert werden, aber auch externe Inhalte.

[28] Bestimmte Teile der Anwendung ein-/ausblenden, Suchergebnisse bis zum nächsten Besuch persistent speichern, bei neu hinzukommenden Daten im Interessengebiet des Nutzers an diesen automatisch eine E-Mail mit dem Link zu der Ressource senden usw.

Möglichkeit der persistenten Registrierung und die Realisierung aller sinnvollen, dadurch möglich werdenden Dienste, die zu einer Personalisierung des Angebots führen.

3.2.2. Vorgehensweise, Priorisierung

Dieser Abschnitt soll die eben beschriebenen Architekturvisionen hinsichtlich ihrer Realisierbarkeit im Rahmen des Diploms und ihrer Kosten-Nutzen-Relation analysieren. Ziel ist die Zuordnung von Prioritäten und damit die Definition einer Reihenfolge, in der die einzelnen Ziele gegebenenfalls anzugehen sind. Dabei ist zu beachten, dass alle Architekturvisionen grundsätzlich sinnvoll sind, aber aus Kosten-/Zeitgründen eine Auswahl vordergründig zu realisierender Aspekte getroffen werden muss. Alle Visionen, die während des Diploms nicht berücksichtigt werden können, sollten in einer nächsten Ausbaustufe des HSSS wieder in Betracht gezogen werden.

Die erste Architekturvision, die eine XML-basierte Struktur für alle temporär persistenten Metadatensätze verlangt, ist mit relativ wenig Aufwand realisierbar und deshalb bereits im Rahmen der Diplomarbeit zu unterstützen. Dazu ist das Java Standard Development Kit (JSDK) um entsprechende Bibliotheken zu ergänzen, die eine Behandlung von XML in Java erlauben. Solche Bibliotheken werden von verschiedenen Herstellern angeboten und stehen in der Regel sogar kostenlos und im Quelltext zur Verfügung. Die abzudeckenden Funktionalitäten sind dabei das Einlesen und Auswerten von XML-Strukturen (realisierbar durch einen XML-Parser) sowie das Erstellen von XML-Strukturen. Unter Umständen könnte auch ein API zur Transformation von XML (XSL/XSLT/FO-Unterstützung) für die Gesamtanwendung von Interesse sein.

Die zweite Architekturvision verlangt die Entkopplung aller sich auf Metadaten beziehenden Informationen aus der Anwendung in eine zu definierende XML-Struktur als Ausgangspunkt für die automatisierte Generierung aller von diesen Informationen abhängenden dynamischen Teile der Anwendung. Eine volle Unterstützung dieser Anforderung ist mit einer zusätzlichen Komplexität von erheblichem Ausmaß und damit auch mit einem enormen Zeitaufwand verbunden. Da die XML-Struktur ebenfalls die Generierung der vom Metadatensatz abhängigen Geschäftsobjekte und deren Struktur beschreiben müsste, ist eine sehr komplexe Ausgangsstruktur von großer Mächtigkeit notwendig, in die sehr viel investiert werden müsste. Zusätzlich muss diese Komplexität für den Anwender entsprechend verborgen werden, er soll ja gerade in der Lage sein, die dynamischen Teile der Anwendung relativ intuitiv und ohne besondere Programmierkenntnisse zu erstellen bzw. zu ändern. Dies wird höchstwahrscheinlich ein

weiteres, ebenfalls sehr komplexes Werkzeug zur Entwicklung / Änderung von Metadatensätzen notwendig machen. Die Erstellung von Anwendungsteilen zur Laufzeit verlangt weiterhin zusätzliche, zum Teil sehr aufwendige Strategien zum Management[29] dieser Vorgänge. Deshalb muss diese Architekturvision für das Diplom etwas eingeschränkt werden. Es sollen nur die Teile realisiert werden, die sich mit relativ wenig Aufwand entwickeln lassen, dabei aber einen großen Nutzen für die Gesamtanwendung versprechen. Dazu gehört auf jeden Fall die strukturelle Beschreibung von Metadatensätzen in XML. Zusätzlich soll diese XML-basierte Metadatensatzbeschreibung Parameter enthalten, die zwar keine automatisierte Erstellung von Anwendungsteilen oder Persistenzstrukturen ermöglichen, aber statisch vorhandene Klassen von außen entsprechend modifizieren können, so dass eine Funktionalität in Abhängigkeit des Metadatensatzes entsteht. Solche Parameter wurden in den Architekturvisionen zum Teil beschrieben und könnten insbesondere die grafische Benutzerschnittstelle und die Validator-Klassen beeinflussen. Alle weiteren Möglichkeiten sollten zu einem späteren Zeitpunkt wieder in Betracht gezogen werden, unter Umständen auch von einer anderen Institution als der SLUB[30]. Dabei muss noch einmal hervorgehoben werden, dass die volle Realisierung der zweiten Architekturvision im Hinblick auf eine automatisierte Erstellung dynamischer Programmteile und Arbeitsabläufe eine zusätzliche Schnittstelle benötigt, um die entstehende Komplexität in den Konfigurationsstrukturen vor den Nutzern zu verbergen. Die Realisierung einer solchen Schnittstelle erfordert unter Umständen enorme Investitionen, so dass die Frage gestellt werden muss, ob sich ein solcher Aufwand lohnt, oder ob das Komplexitätsproblem nicht nur auf eine andere Abstraktionsebene verschoben wird.

Architekturvision drei und vier beziehen sich auf den hauptsächlich in der Diplomarbeit zu realisierenden Anwendungsteil, zu dem noch keine Vorarbeit aus der Belegarbeit existiert. Gegebenenfalls müssen deshalb auch hier Einschränkungen getroffen werden, wenn der zu erwartende Aufwand die Möglichkeiten einer Diplomarbeit zu überschreiten droht. Die dritte Architekturvision, die eine dynamische, durch

[29] So zum Beispiel Cache-Strategien, so dass die Anwendung entscheiden kann, ob neue Objekte erstellt werden müssen bzw. ob schon vorhandene Objekte lediglich verändert zu werden brauchen und vieles mehr

[30] Die volle Umsetzung der zweiten Architekturvision in Verbindung mit der zusätzlichen Möglichkeit, nach demselben Muster einen (relativ) beliebigen Arbeitsablauf für die Behandlung der Metadaten zu definieren, führt zwangsläufig zu einer hochgradig generischen und flexiblen Anwendung, die einen Großteil der Funktionalität eines Hochschulschriftenservers realisiert und aufgrund der Kombination von Java und XML an Plattformunabhängigkeit kaum zu übertreffen sein dürfte. Insofern wäre sie ideal geeignet, für beliebige Institutionen als „Plug and Play“-Lösung zu dienen, die an einer technisch fortgeschrittenen und vor allem flexibel konfigurierbaren Lösung interessiert sind. Eine solche Anwendung würde sich natürlich ideal als Referenzimplementierung für eine Orgaisation eignen, welche die nationale Koordinierung von elektronischer Publizistik im Wissenschaftsbereich übernimmt, so wie sie in Abschnitt 2.4.4 auf Seite 56 gefordert wurde.

den Nutzer bestimmbare Skalierung der Auflösung der Web-Schnittstelle verlangt, soll allerdings vollständig unterstützt werden. Probleme könnten sich vor allem durch die ebenfalls notwendige Skalierung von Grafiken in Verbindung mit der schlechten Unterstützung von vektorbasierten Formaten ergeben, gegebenenfalls muss hier die dynamische Skalierung auf eine Skalierung für eine begrenzte Zahl von vorgegebenen Auflösungen reduziert werden. Die vierte Vision, die eine Portal-basierte und stark personalisierte Umsetzung der dritten Teilanwendung verlangt, wird aufgrund der sich daraus ergebenen hohen Entwicklungskosten nur teilweise umgesetzt werden können. Insbesondere die persistente Registrierung unter Nutzung der Datenbank wird einer späteren Ausbaustufe vorbehalten bleiben müssen. Eine solche Ausbaustufe soll aber durch eine Session-Unterstützung vorbereitet werden. Dies bedeutet gleichzeitig, dass Informationen zumindest über den Zeitraum der Sitzung aufrecht erhalten werden können, was bereits die Realisierung einer Reihe von nützlichen Diensten ermöglicht. So ließen sich zum Beispiel vom Nutzer verschiedene interessante „Treffer“ seiner Suchen “sammeln“ und könnten ihm später zusammenhängend präsentiert werden. Eine spätere Ausbaustufe könnte solche Sammlungen sogar über die Sitzung hinaus für den Nutzer persistent vorhalten.

3.3. Architekturdetails

Nach dem bis hierher die Gesamtarchitektur auf einem relativ hohen Abstraktionsniveau und außerdem sehr generisch betrachtete wurde, sollen in diesem Abschnitt konkrete Architekturdetails der Gesamtanwendung[31] betrachtet werden. Dazu wird die Gesamtanwendung logisch in eine Server-Ebene, eine Client-Ebene und eine Datenbank-Ebene nach dem Drei-Schichten-Modell von Client-Server-Anwendungen unterteilt, welches **Abbildung 18** zeigt:

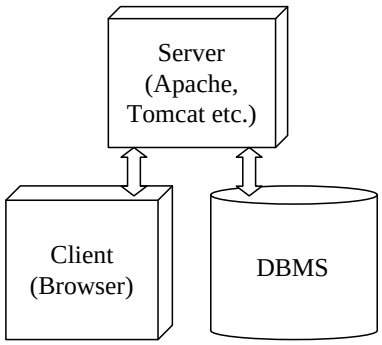

Abbildung 18 *- Drei-Schichten-Modell*

Die Modellabbildung zeigt die drei zu separierenden Schichten Client, Server und Datenbank, die sich in der Regel auch auf physisch getrennten Rechnern befinden. Diese Einteilung der Anwendung ist allerdings nicht ganz unproblematisch. So gehören die Java ServerPages, die im Abschnitt der Client-Ebene besprochen werden, strenggenommen auch zu den Server-seitigen Anwendungen[32]. Da die Gesamtanwendung aber konsequent das Model-View-Controller-Paradigma für Web-Anwendungen realisiert (siehe [Sie00b]) und die JSP's für den View-Bereich verantwortlich sind, der wiederum die Schnittstelle des Clients definiert, ist diese Einteilung vertretbar. Zusätzlich werden in dem sich anschließenden Abschnitt noch einige Besonderheiten der Architektur betrachtet, wie zum Beispiel die Konfigurationsstrukturen der Gesamtanwendung.

[31] Auch die Architekturdetails beziehen sich auf die gesamte Anwendung, die allerdings nach gewissen Gesichtspunkten unterteilt wird. Architekturdetails der einzelnen Teilanwendungen werden hingegen in den folgenden Kapiteln beschrieben.

[32] Da JSP's nicht anderes als eine besondere Form eines Servlets sind.

3.3.1. Architekturbesonderheiten

In diesem Abschnitt sollen einige Architekturbesonderheiten vorgestellt werden, welche die gesamte Anwendung betreffen und deren Kenntnis für das Verständnis der im weiteren Verlauf der Diplomarbeit gezeigten Entwurfsentscheidungen von Vorteil ist. Dabei handelt es sich zum einen um den Algorithmus zur Erstellung einer eindeutigen Bearbeitungsnummer für die Metadatensätze und um die Vorstellung der verschiedenen Konfigurationsstrukturen, durch die die Anwendung von außen weitestgehend konfigurierbar ist.

3.3.1.1. Bearbeitungsnummer

Für die Behandlung der Metadatensätze benötigt die Anwendung eine eindeutige Bearbeitungsnummer, um diese unterscheiden zu können. Eine solche Bearbeitungsnummer ist nicht nur während der Bearbeitung der Metadatensätze sinnvoll, sondern kann auch für viele andere Zwecke genutzt werden, wo die eindeutige Unterscheidung der Metadatensätze notwendig ist, so zum Beispiel zur Erstellung einer eindeutigen Verzeichnisstruktur oder zur Generierung der absoluten URL. Auch für Exportinformationen kann diese Bearbeitungsnummer verwendet werden, da die externen Institutionen in der Regel ebenfalls eine Identifikation benötigen, die innerhalb des jeweiligen Archivs eindeutig ist.

Eine Bearbeitungsnummer innerhalb des HSSS setzt sich aus zwei Teilen zusammen. Zum einen wird ein Zeitstempel benutzt, wie er von vielen Systemen her bekannt ist. Dieser besteht aus den Millisekunden, die seit dem 01. Januar 1970 um 00:00 Uhr GMT vergangen sind. Dieser Wert ist sehr einfach zu erstellen und relativ eindeutig und hat zusätzlich den Vorteil, dass er eine verwertbare Information in sich trägt[33]. Allerdings besteht bei einem verteilten System wie dem HSSS die theoretische Möglichkeit, dass sich solche Bearbeitungsnummern gleichen, wenn sie zufällig zur selben Zeit erstellt wurden oder unglücklicherweise in dem Zeitabstand, der dem Unterschied der beiden lokalen Systemzeiten entspricht. Diese Wahrscheinlichkeit soll durch die Integration eines zweiten Bestandteils in die Bearbeitungsnummer verringert werden. Dabei handelt es sich um eine sichere, 4-stellige Zufallszahl[LFB00][34], die mit einem

[33] der Zeitpunkt der Initialisierung der Bearbeitungsnummer durch das System

[34] Dabei wird nicht der normale Zufallszahlengenerator aus dem Package `java.util` verwendet, weil dieser keine kryptographisch zufälligen Werte erzeugen kann, sondern ein entsprechender Genera-

Trennzeichen an den Zeitstempel angefügt wird. Zusammengesetzt hat eine Bearbeitungsnummer also die Form

`<timestamp>-<random>`, z. Bsp. `994313565203-3414`

Die Wahrscheinlichkeit einer Kollision ist damit äußerst gering und würde überdies im weiteren Verlauf durch die Anwendung erkannt werden. Der große Vorteil dieses Ansatzes ist der. dass die Anwendung zur Erstellung der Bearbeitungsnummer keine weiteren externen Informationen benötigt und daher sehr unabhängig ist.

Alternativ wäre natürlich auch ein inkrementeller Ansatz denkbar, der zunächst auch trivialer anmutet. Allerdings bliebe in diesem Fall die Verantwortung, eindeutige Bearbeitungsnummern durch das Inkrementieren eines bestimmten Wertes zu erzielen, direkt bei der Anwendung. Diese ist dazu auf die persistente Information der letzten vergebenen Bearbeitungsnummer angewiesen, was ein Nachteil bei der Erstellung ist, sowohl was die Performance als auch die Unabhängigkeit betrifft.

tor aus dem Package `java.security`, welcher sichere Zufallszahlen generiert.

3.3.1.2. Konfigurationsstrukturen

Wie aus den Anforderungen an den HSSS und insbesondere aus den Architekturvisionen hervorgegangen ist, soll sich die Gesamtanwendung vor allem durch eine hohe Flexibilität und eine weitgehende Konfigurierbarkeit auszeichnen. So soll u.a. die Metadatensatzbeschreibung XML-basiert vorliegen, um bestimmte Verhaltensweisen der damit zusammenhängenden Anwendungsteile von außen beeinflussen zu können. Eine solche externe Konfigurierbarkeit ist selbstverständlich auch für andere Anwendungsteile von großem Vorteil. Deshalb liegt die Vorstellung nahe, generell und so weit wie möglich auf XML-basierte Konfigurationsstrukturen zu setzen, die dann allerdings einer gewissen Aufteilung bedürfen. Mit der Architektur dieser Konfigurationsstrukturen und deren logischer Separierung soll sich dieser Abschnitt im weiteren Verlauf beschäftigen. Die Verwendung von XML-basierten Konfigurationsdateien wird von der gewählten Java-basierten Servertechnologie[35] sogar bereits unterstützt. Die Servlet-Spezifikation von Sun [DDC99] verlangt die Existenz einer „`web.xml`“ zu benennenden Konfigurationsdatei für jede Web-Anwendung. Der Inhalt dieser Konfigurationsdatei gehorcht einer bestimmten Dokumententypdefinition (DTD) und ist von dem entsprechenden Servlet-Container bei der Initialisierung der Web-Anwendung einzulesen und in einem bestimmten, dem Servlet verfügbaren Objekt (`javax.servlet.ServletConfig`) abzulegen. **Listing 6** zeigt die grundlegende Struktur dieser Konfigurationsdatei:

```
1  <?xml version="1.0" encoding="ISO−8859−1" ?>
2  <!DOCTYPE web−app PUBLIC
3     "−//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
4     "http://java.sun.com/j2ee/dtds/web−app_2.2.dtd">
5  <web−app>
6     <servlet>
7        <servlet−name>MetadataGeneratorServlet</servlet−name>
8        <servlet−class>hsss.metadatagenerator.MetadataGeneratorServlet</servlet−class>
9        <init−param>
10          <param−name>servlet.version</param−name>
11          <param−value>1.5b3</param−value>
12       </init−param>
13       <init−param>
14          <param−name>file.hsssdata.xml</param−name>
15          <param−value>d:/jakarta−tomcat−3.2/hsss/xml/hsssdata.xml</param−value>
16       </init−param>
17       ...
18    </servlet>
19    <servlet>
20       ...
21    </servlet>
22    ... usw.
23 </web−app>
```

***Listing 6** : grundlegende Struktur der Konfigurationsdatei „web.xml“*

[35] weitere Informationen zu Architektur-Details der Server-Ebene finden sich im anschließenden Abschnitt 3.3.2

Dabei wird deutlich, dass sich eine solche Konfigurationsdatei auf genau eine Web-Anwendung bezieht (**<web-app>** für Web Application), welche wiederum aus einer beliebigen Anzahl von Servlets bestehen kann, die jeweils einzeln konfigurierbar sind (zwischen den **<servlet>**-Tags). Die Möglichkeiten dieser Konfigurationsdatei sind allerdings sehr eingeschränkt, da sich Informationen an die Anwendung nur in Form von Parameter-Wert-Paaren (**<param-name>** bzw. **<param-value>** jeweils innerhalb eines **<init-param>**-Tags) weitergeben lassen. Es macht daher Sinn, nur die fundamentalsten bzw. relativ unstrukturierte Konfigurationsparameter auf diese Weise direkt an die Anwendung zu übermitteln und weitere Informationen in separaten XML-Strukturen zur Verfügung zu stellen, deren Lokation der Anwendung allerdings bekannt gemacht werden muss (siehe Zeilen 13 bis 16 im Listing). In diesem Fall wird eine Lokation im Dateisystem angegeben, genausogut könnte aber eine URL oder eine Datenbankverbindung angegeben werden. Auf diese Art und Weise wird die Gesamtanwendung des HSSS über die Lokation der verschiedenen Konfigurationsstrukturen[36] informiert, die in **Tabelle 4** näher erläutert werden[37]:

Konfigurationsstruktur	Inhaltsbeschreibung
web.xml	Enthält (wie bereits beschrieben) die allgemeinen Konfigurationsparameter der Web-Application „HSSS“ in XML-basierter Form.
hsssdata.xml	Enthält Konfigurationsdaten, die den gesamten Hochschulschriftenserver HSSS betreffen, wie z. Bsp unterstützte Sprachen, Formate usw. (in XML-basierter Form).
MetadataSetDissertation.xml	Enthält die XML-basierte Metadatensatzbeschreibung für die Dokumentenart „Dissertation“.
MetadataSetBericht.xml	Enthält die XML-basierte Metadatensatzbeschreibung für die Dokumentenart „Bericht“.
oaisets.xml	Enthält die von der OAI-Implementierung des HSSS unterstützten *Set*-Strukturen in XML-basierter Form.
hsssdb.props	Enthält Konfigurationsdaten für das DBMS (nicht XML-basiert).

***Tabelle 4** - Konfigurationsstrukturen des HSSS*

[36] Es ist absichtlich nicht von Konfigurationsdateien die Rede, weil die entsprechenden XML-Strukturen nicht notwendigerweise in Dateiform vorliegen müssen, sondern sich zum Beispiel auch in einer Datenbank befinden können.

[37] Weitergehende Informationen finden sich in Abschnitt B im Anhang ab Seite 250 bzw. in [JMS01]

Die einzelnen Teilstrukturen werden an geeigneten Stellen (z. Bsp. in den Kapiteln zu den Teilanwendungen) im weiteren Verlauf der Diplomarbeit vertieft, außerdem finden sich Informationen über die jeweiligen Inhalte im Anhang (Kapitel B ab Seite 250) sowie eine detaillierte Beschreibung aller Parameter in [JMS01].

3.3.2. Architekturdetails auf Server-Ebene

Dieser Abschnitt, der sich mit einigen Architekturdetails auf der Server-Ebene der Gesamtanwendung beschäftigen soll, wird zunächst kurz die Infrastruktur bzw. die Technologieplattform vorstellen, die als Grundlage für die Server-seitigen Teile der Anwendung dient, und im Anschluß auf mögliche Alternativen eingehen. Da diese Thematik sehr ausführlich im Rahmen der Belegarbeit abgehandelt wurde und sich zudem in diesem Bereich so gut wie keine Änderungen ergaben, ist der Umfang der Ausführungen relativ gering, wobei aber auf die Belegarbeit als zusätzliche Informationsquelle verwiesen werden kann[Schb00]. Dort wurde bereits eine begründete Technologieentscheidung zugunsten einer Java-basierten Plattform (Servlets, Java ServerPages) getroffen. An dieser Entscheidung, die durch die im Beleg entstandenen Ergebnisse auch absolut bestätigt werden konnte, wird im Diplom festgehalten. Auch an den Komponenten Webserver (*Apache*) und Servlet-Container (*Tomcat*) werden keine Veränderungen vorgenommen. Insgesamt ergibt sich das in der aus [Schb00] entliehenen **Abbildung 19** gezeigte Zusammenspiel dieser Komponenten als Infrastruktur für die Gesamtanwendung:

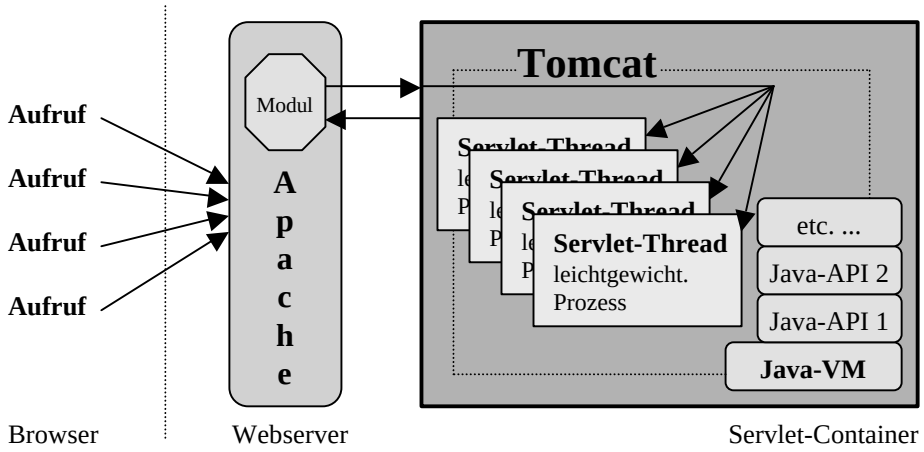

Abbildung 19 *- An der Gesamtanwendung beteiligte Komponenten*

Der *Apache* Webserver leitet die Nutzeranfragen für die jeweiligen Webanwendungen über ein spezielles Modul[38] an den Servlet-Container *Tomcat* weiter. Dieser ist in der Lage, diese Anfragen zu beantworten, wobei die Abbildung noch einmal verdeutlicht, dass *Tomcat* für jede (parallele) Anfrage lediglich einen neuen Thread erzeugt, was insbesondere bei großen Zugriffsraten mit hoher Parallelität zu einer deutlich besseren Performance und Stabilität führt im Vergleich zu Technologiealternativen, die jeweils einen neuen Prozess erzeugen. Innerhalb des Servlet-Containers übernimmt das jeweilige Servlet der Teilanwendung die Kontrolle über die Anfrage und nutzt die Java-basierte Gesamtanwendung, welche wiederum die Java Standard-API's, aber auch eine Reihe zusätzlicher Bibliotheken nutzt. In dieser Abbildung fehlt noch die Datenbankschicht, auf die ebenfalls aus der Servletumgebung heraus über die Java-Anwendung mit Hilfe eines entsprechenden JDBC-Treibers zugegriffen wird[39].

3.3.2.1. Technologische Alternativen

In der Belegarbeit wurden die technischen Alternativen zur eben vorgestellten Java-basierten Infrastruktur ausgiebig analysiert und beschrieben[Schb00]. An dieser Stelle sollen deshalb die vorhandenen Alternativen nur kurz genannt und die wichtigsten Vor- und Nachteile im Vergleich zur gewählten Architektur zusammengefasst werden:

- **Common Gateway Interface (CGI)**
  Hierbei handelt es sich um die älteste und zur Zeit wohl noch verbreitetste Technologie, um Webanwendungen zu realisieren. CGI definiert eine Schnittstelle zwischen Webserver und externen Programmen, die über diese Schnittstelle in den Prozess der Anfragebearbeitung mit einbezogen werden.
  - Vorteile:
    Nahezu jeder Webserver unterstützt diese Technik und nahezu jede Programmiersprache lässt sich über CGI integrieren[40].
  - Nachteile:
    Einer der gravierendsten Nachteile dieser Technologie ist der, dass für jeden Zugriff ein neuer, externer (im Sinne von außerhalb des Webservers) Prozess gestartet wird, was zu einem instabilen Systemverhalten bei hohen Zugriffsraten mit dementsprechend schlechter Performance führt. Außerdem ist die Aufrufsementik hinter CGI antiquiert und undurchsichtig, so dass sich diese Technologie (falls überhaupt) allenfalls für kleine bis mittlere Webanwendungen eignet.

[38] Apache Modul modjk
[39] Details zu diesem Thema finden sich im Abschnitt 3.3.4 ab Seite 113
[40] Am Häufigsten ist die Integration von Perl verbreitet

- **CGI-Erweiterungen**
  Hierbei handelt es sich um propritäre Erweiterungen von CGI, welche die beschriebenen Nachteile von CGI mildern sollen[41].

  – Vorteile:
  Die beschriebenen CGI-Nachteile können etwas abgemildert werden, insbesondere was das Prozessverhalten bei parallelen Zugriffen betrifft.

  – Nachteile:
  Diese Erweiterungen sind in der Regel auf einen bestimmten Webserver zugeschnitten und zum Teil auch plattformabhängig. Dadurch besteht die Gefahr einer gewissen Abhängigkeit vom entsprechenden Hersteller.

- **Active Server Pages (ASP)**
  Active Server Pages (ASP) wurden von Microsoft entwickelt und beinhalten eine Kombination von HTML-Teilen und speziellen ASP-Teilen im Quellcode. Außerdem erlauben sie die Integration externer Objekte über Techniken wie (u.a.) COM, DCOM und OLE.

  – Vorteile:
  ASP stellt ein flexibles Konzept zur Erstellung Server-seitiger Webanwendungen dar, welches sich außerdem dadurch auszeichnet, dass theoretisch bis zu einem gewissen Grade eine Trennung von Layout und Applikationslogik möglich ist. Nicht umsonst besteht bis zu einem gewissen Grad eine Ähnlichkeit zwischen ASP und JSP.

  – Nachteile:
  Das große Manko dieser Technologie ist ihre Plattformabhängigkeit. So ist sie zum einen an die Win32-Plattformen gebunden (will man externe ActiveX-Objecte verwenden) und auch an den Webserver (Microsoft Internet Information Server). Es besteht also in allen Teilen eine sehr große Abhängigkeit zum Hersteller Microsoft.

- **PHP Hypertext Preprocessor (PHP)**
  PHP ermöglicht ebenfalls die Integration von HTML- und PHP-Code in eine Quelle und hat sich insbesondere bei der Entwicklung kleinerer bis mittelgroßer, Server-seitig dynamischer Web-Angebote durchgesetzt. Gerade in Verbindung mit anderer Open Source-Technologie wie *Apache* und *MySQL* ist der Einsatz von PHP weit verbreitet.

  – Vorteile:
  PHP ist Open-Source-Software, es bestehen somit kaum Abhängigkeiten.

[41] Die bekanntesten Vertreter sind ISAPI (Internet Server Application Programming Interface) von Microsoft sowie NSAPI (Netscape Server Application Programming Interface) von Netscape

PHP wird intensiv weiterentwickelt, es existieren außerdem eine Vielzahl von API's für fast jede Funktionalität und eine große Zahl von Webanwendungen, die auf dieser Technologie basieren.

– Nachteile:
PHP eignet sich zur Zeit noch nicht uneingeschränkt für die Entwicklung großer und verteilter Webanwendungen. Architekturkonzepte, wie sie beispielsweise Java seit Jahren unterstützt, halten im PHP-Bereich erst in letzter Zeit Einzug und erreichen bei weitem nicht die erforderliche Integrität.

Die wichtigsten Vorteile der gewählten Technologie, die auf Java basiert und aus eine Kombination von Servlets und Java ServerPages besteht, werden im Folgenden noch einmal zusammengefasst:

- Die gewählte Technologie zeichnet sich durch konsequente Objektorientierung, intuitive Aufrufsemantiken und offene Spezifikationen aus.
- Sie ist aufgrund ihrer Architektur weitestgehend plattformunabhängig.
- Im Vergleich zu den meisten Alternativen zeichnet sich die gewählte Lösung durch eine höhere Stabilität im Laufzeitverhalten und eine gute Performance auch in Lastsituationen aus.
- Die Technologie basiert auf einer Programmiersprache, die zu den modernsten Sprachen zählt und sich nach wie vor überdurchschnittlich erfolgreich entwickelt. Somit stehen eine Vielzahl von Schnittstellen und Bibliotheken in der Regel zur freien Verfügung.
- Die Unterstützung dieser Technologie durch die Webserver nimmt ständig zu. Durch starke Präsenz dieser Technologie im Enterprise Application Bereich (J2EE) führt in Zukunft kein Weg an ihr vorbei.
- Im Vergleich zu allen Alternativen besitzt die gewählte Technologie eine Reihe von Alleinstellungsmerkmalen, die sich aus einem technologischen und konzeptionellen Vorsprung ergeben.

3.3.2.2. Objektorientierte Analyse und Design

Schon im Beleg wurden Entwurf und Implementation des Gesamtsystems nach objektorientierten Prinzipien gestaltet, also insbesondere durch objektorientierte Analyse

bzw. Design vorbereitet. Selbstverständlich wurde diese Vorgehensweise auch während des Diploms beibehalten. Ein erster Schritt bestand dabei in der genauen Analyse und der Spezifikation der Geschäftsmodelle und aller damit in Zusammenhang stehenden Arbeitsabläufe, was bereits ausführlich in Abschnitt 3.1 ab Seite 62 beschrieben wurde. Die hier gewonnenen Informationen wurden als Ausgangspunkt für alle weiteren objektorientierten Analysen verwendet, die das Design und die Architektur des Gesamtsystems betrafen[42]. Dabei wurde die bereits in der Belegarbeit aufgestellte Maxime beibehalten, die Gesamtanwendung nach logischen Gesichtspunkten zu separieren. Dies wurde durch eine entsprechende Package-Struktur[43] der Gesamtanwendung in Java realisiert. **Abbildung 20** zeigt diese Package-Struktur, welche die Gesamtanwendung in einer sehr groben Granularität strukturiert bzw. schichtet (siehe dazu auch den folgenden Abschnitt 3.3.2.3 ab Seite 105).

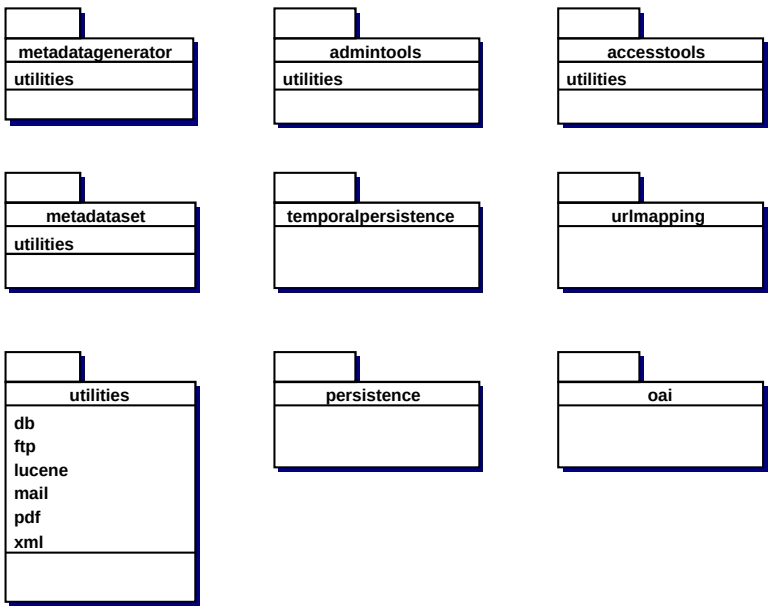

***Abbildung 20** - HSSS Package-Struktur*

Die Abbildung zeigt die Packages innerhalb des Package „`hsss`“, welches das oberste in der Package-Hierarchie der Gesamtanwendung ist (die Package-Struktur entspricht auf Dateisystem-Ebene der Verzeichnishierarchie, welche zum besseren Verständnis der Struktur in **Abbildung 21** gezeigt wird). Zu erkennen ist, dass jeweils ein separates Package für die Teilanwendungen *MetadatenGenerator*, *AdminTools* und

[42] Details zur objektorientierten Analyse und Design der Teilanwendungen finden sich in den die entsprechende Teilanwendung jeweils speziell betreffenden Kapiteln

[43] Ein Package (Paket) bezeichnet in Java eine Menge von Klassen, die zueinander in einer besonderen (engen) Beziehung stehen.

AccessTools existiert („`metadatagenerator`“, „`admintools`“ und „`accesstools`“) sowie ein Package („`oai`“) für die Implementation des OAI-Protokolls. Innerhalb der Packages sind gegebenenfalls auch die Namen eventuell enthaltender Packages sichtbar. Die Packages der Teilanwendungen zum Beispiel enthalten jeweils ein Package namens „`utilities`“, welches spezifisch auf die Teilanwendung zugeschnittene Hilfsklassen enthält. Daneben existieren mit „`metadatset`“, „`temporalpersistence`“, „`persistence`“ und „`utilities`“ weitere Packages, die für die gesamte Anwendung von Bedeutung sind, sich also nicht einzelnen Teilanwendungen explizit zuordnen lassen. Dabei enthält das Package „`metadataset`“ alle Klassen, welche die Umsetzung der Metadatensatzstruktur in die entsprechende Struktur von Java-Objekten realisieren (also die eigentlichen „Geschäftsobjekte“) sowie Klassen, die sich inhaltlich auf den Metadatensatz beziehen (Validatorklassen etc.). Auch dieses Package besitzt eine Reihe von Werkzeugklassen in einem separaten Package „`utilities`“. Die Packages „`persistence`“ und „`temporalpersistence`“ enthalten Klassen, die sich schwerpunktmäßig im weitesten Sinne mit Persistenz (dazu gehört auch das Einlesen und Schreiben von Konfigurationsdateien) bzw. temporärer Persistenz (das XML-basierte Zwischenspeichern der Metadatensätze während ihrer Bearbeitung) beschäftigen. Mit „`urlmapping`“ ist ein Package bezeichnet, dessen Klassen die Abbildung der absoluten (virtuellen) URL's auf die zugehörigen realen URL's vornehmen. Das Package „`utilities`“ schließlich enthält Klassen, die in der gesamten Anwendung Verwendung finden (zum Teil auch einfache Geschäftsobjekte) sowie eine Reihe von Hilfs- und Werkzeugklassen. Außerdem integriert bzw. kapselt dieses Package in jeweils separaten Packages den Zugriff auf externe API's (in den Packages „`db`“, „`ftp`“, „`lucene`“, „`mail`“ und „`xml`“).

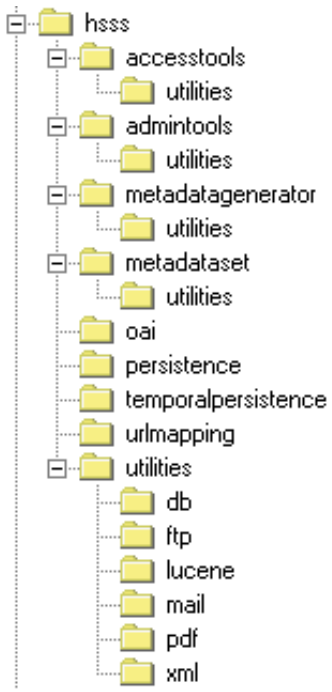

***Abbildung 21** - HSSS Verzeichnishierarchie*

Im folgenden sollen nur die Packages in ihrer Struktur und Funktion ein wenig näher betrachtet werden, die sich auf die gesamte Anwendung beziehen, die Packages der Teilanwendungen werden später in separaten Kapiteln beschrieben. **Abbildung 22** auf der folgenden Seite zeigt das Klassendiagramm des Packages „`persistence`“ (ausschnittsweise). Dieses Package dient insbesondere der Interaktion mit dem Teil der Konfigurationsstrukturen, der in „`hsssdata.xml`“ enthalten ist (siehe auch Abschnitt B.3 im Anhang). Wie aus der Abbildung ersichtlich wird, werden zwei Persistenzarten

unterstützt, d.h. die Konfigurationsstruktur kann sowohl dateibasiert als auch datenbankbasiert vorliegen und wird von einer jeweils spezialisierten Klasse[44] entsprechend behandelt (eingelesen) und steht danach der entsprechenden Teilanwendung über Methoden wie `getSupportedLanguages()` (alle vom HSSS unterstützten Sprachen) oder `getSupportedFormats()` (alle vom HSSS unterstützten Dokumentenformate) usw. zur Verfügung.

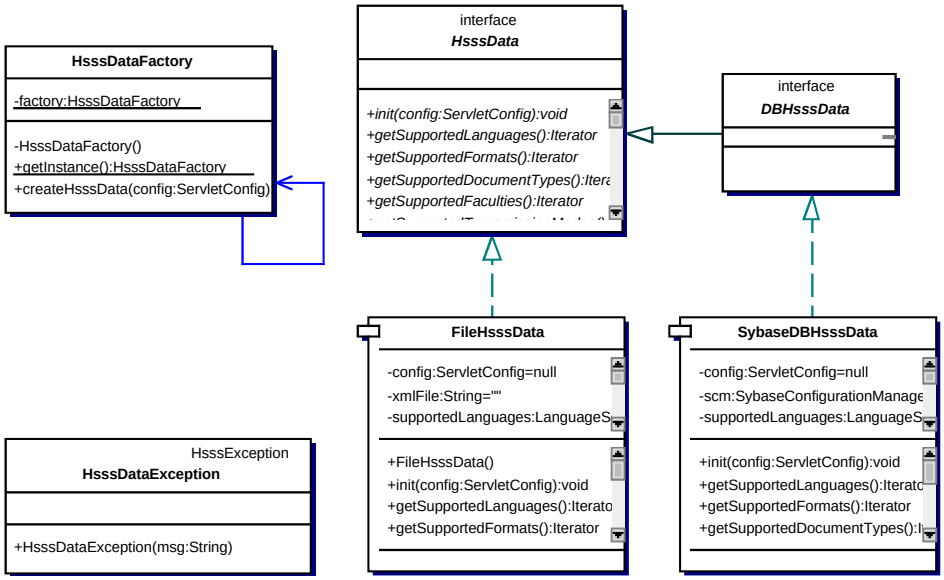

***Abbildung 22** - HSSS Package "persistence" (Ausschnitt)*

Das Package **„`temporalpersistence`“**, welches im folgenden Abschnitt strukturell näher erläutert wird, ist für die Umwandlung der Metadatensatz-Repräsentation von einer Objektstruktur in eine XML-basierte Struktur (und umgekehrt) zuständig (siehe Architekturvision 1 in Abschnitt 3.2.1) sowie für verschiedene weitere Aktionen im Zusammenhang mit den Metadatensätzen (Löschen, Rücksetzen, Einstellen usw.). Das Package **„`utilities`“** ist das umfangreichste der Gesamtanwendung und besteht neben einer Reihe von weiteren Packages vor allem aus Geschäftsobjekten, Hilfs- und Systemklassen, wie **Abbildung 23** auf der folgenden Seite verdeutlichen soll. Am Beispiel des Geschäftsobjekts „Universität“ soll der Aufbau vieler ähnlich konstruierten Strukturen innerhalb dieses Packages demonstriert werden. Das Geschäftsobjekt „Universität“ wird beispielsweise innerhalb des *Dublin Core*-Elements „DC.Contributor“ benötigt und wird hier durch die Klasse `UniversityInstitution` repräsentiert, welche ihrerseits von der Klasse `Institution` abgeleitet ist und zusätzlich drei Aggregate diesen Typs enthält (nämlich `department` (Fachbereich), `faculty`

[44] Nähere Informationen zu dieser Klassenstruktur liefert der sich anschließende Abschnitt 3.3.2.3 ab Seite 105

(Fakultät) und `institute` (Institut)). Außerdem enthält jede Institution ein Aggregat vom Typ `Address` (Adresse), also auch die Universität. Nach diesem objektorientierten Ansatz setzen sich die meisten Geschäftsobjekte aus mehreren primitiven Klassen zusammen, die somit aber auch für andere Zwecke wiederverwendbar bleiben.

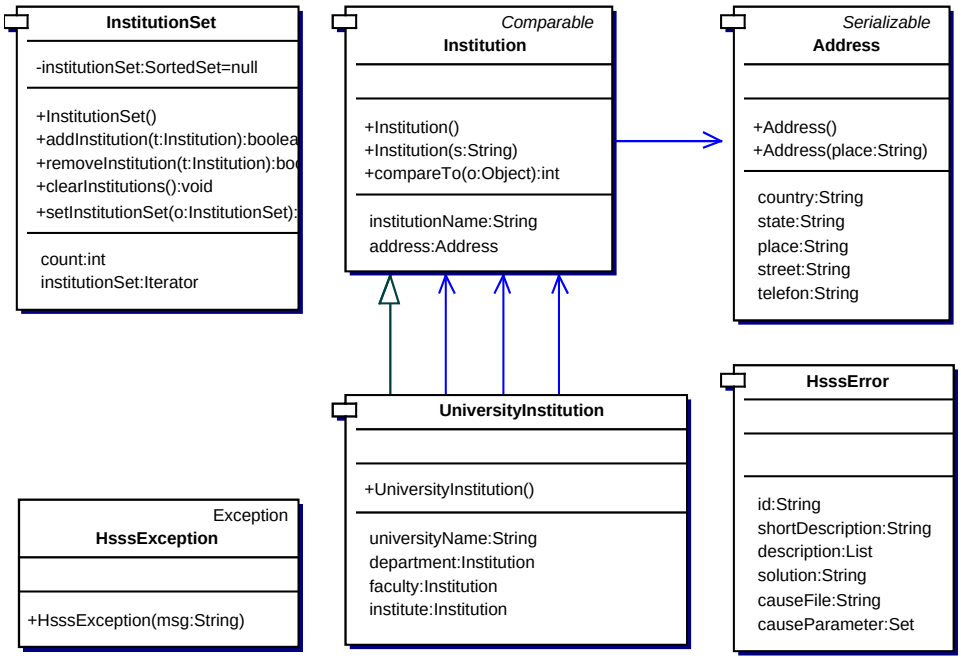

Abbildung 23 *- HSSS Package "utilities" (Ausschnitt)*

Die Klassen `HsssException` und `HsssError` stehen beispielhaft für die ebenfalls in diesem Package zahlreich vorhandenen System-, Hilfs- und Werkzeugklassen. So wird `HsssError` von den Teilanwendungen dafür verwendet, aufgetretene Ausnahmen und Laufzeitfehler zu beschreiben und dem System sowie den Anwendern zur Verfügung zu stellen. Zu den Beschreibungsmöglichkeiten dieser Klasse gehören Fehler-Identifikationsnummer, Kurzbeschreibung, detaillierte Fehlerbeschreibung, Lösungsmöglichkeiten, sowie Angabe der JSP-Datei bzw. der Parameter, die bei der Entstehung des Fehlers beteiligt waren. Die Klasse erlaubt somit die Generierung von verständlichen und ausführlichen Rückmeldungen an die Nutzer des Systems.

Daneben enthält „`utilities`" weitere Packages, auf der nächsten Seite zeigt **Abbildung 24** als Beispiel das Package „`xml`", welches alle für die Gesamtanwendung relevanten Klassen kapselt, die sich im weitesten Sinne mit XML auseinandersetzen. Für fast alle Bearbeitungsschritte im Zusammenhang mit XML wird das API „JDOM"[45]

[45] JDOM ist eine Bibliothek, die als Aufsatz auf verschiedene XML-Parser (SAX, DOM) fungiert und dabei eine auf die Sprache Java optimierte Schnittstelle bietet.

verwendet (siehe dazu auch die Beschreibung dieses API im Anhang in Abschnitt D.2.8 ab Seite 280), die Klasse `JDOMParser` fungiert für die Gesamtanwendung als Wrapper und wird bei jedem Vorgang benötigt, der ein Einlesen bzw. Parsen einer XML-Ressource verlangt. Am Ende des Parsens stellt die Klasse `JDOMParser` eine für die Anwendung sehr gut auswertbare Repräsentation der XML-Struktur dar (ein JDOM-Document), aus welcher dann mit Hilfe der Klassen `HSSSDataRetriever`, `MetadataSetDataRetriever` bzw. `AccessDataRetriever` die eigentliche Objekt-Repräsentation erstellt wird.

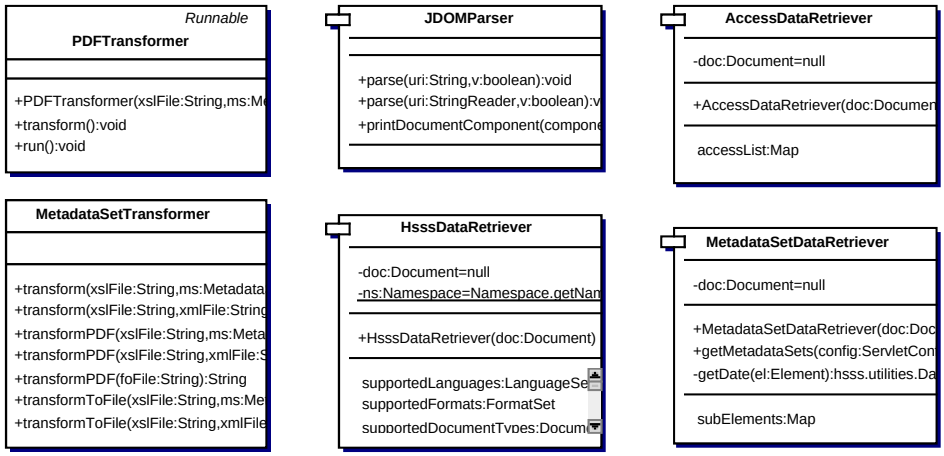

Abbildung 24 *- HSSS Package "xml" (Ausschnitt)*

Die Klasse `HsssDataRetriever` erstellt dabei die Objekt-Repräsentation der Konfigurationsstruktur „`hsssdata.xml`" (unterstützte Sprachen, Formate, Dokumentenarten etc.) und die Klasse `MetadataSetDataRetriever` die Objekt-Repräsentation einer Metadatensatzbeschreibung ("`MetadataSetDissertation.xml`" oder „`MetadataSetBericht.xml`"). Die Klasse `MetadatSetTransformer` kann XML-basierte Metadatensatzstrukturen mit Hilfe von XSL-Dateien und einem XSLT-Prozessor[46] in verschiedene Zielformate (u.a. andere XML-Strukturen, XHTML, PDF usw.) umwandeln. Die Klasse `PDFTransformer` ist auf die Umwandlung von XML in PDF spezialisiert und eignet sich insbesondere auch zum asynchronen Aufruf (um zum Beispiel die PDF-Datei innerhalb eines separaten Threads parallel zu erstellen). Alle weiteren Packages innerhalb von „`utilities`" werden separat im Zusammenspiel mit den Teilanwendungen erläutert.

[46]Die Anwendung verwendet den XSLT-Prozessor Xalan der ASF (siehe Anhang D.2.6 auf Seite 279)

3.3.2.3. Architektur- und Entwurfsmuster

Muster spielen in der modernen Softwareentwicklung eine immer größere Rolle. Mit Hilfe von Mustern lassen sich Erfahrungen im Zusammenhang mit der Analyse, dem Design und der Implementation von Software in Form von Problem-Lösungs-Paaren beschreiben und an andere Entwickler weiter geben. Außerdem erleichtern Muster die Beschreibung komplexer Systeme erheblich, weil sich in ihnen bestimmte Strukturen zusammenfassen und benennen lassen. In der Regel werden Muster zusätzlich in verschiedene Kategorien eingeordnet, die den Abstraktionsgrad des Musters wiederspiegeln (z. Bsp. Architekturmuster, Entwurfsmuster, Paradigmen etc.). Zu diesem Thema sind im Verlauf der letzten Jahre eine Reihe von Publikationen erschienen, auf die an dieser Stelle verwiesen werden soll [GHJ96], [Gra98], [BMR98].

Dieser Abschnitt befasst sich mit Entwurfsmustern und stellt beispielhaft zwei solche Muster vor, die in dem Gesamtsystem relativ häufig Verwendung fanden. Es handelt sich um die Muster *Singleton* und *Fabrikmethode*, die an verschiedenen Stellen der Gesamtanwendung immer wieder eingesetzt wurden. Die folgende Auflistung enthält jeweils eine kurze Definition des Musters (nach [GHJ96] bzw. [Gra98]), die folgenden Abbildungen zeigen jeweils ein formales Klassendiagramm für die Muster.

- ***Singleton:***
  Es wird abgesichert, dass eine Klasse genau ein Exemplar besitzt, und es wird ein globaler Zugriffspunkt bereitgestellt.
- ***Fabrikmethode:***
  Es wird eine Klasse mit einer Operation zur Erzeugung von Objekten in Abhängigkeit eines bestimmten Diskriminators realisiert. Andere Klassen können die Erzeugung von solchen Objekten komplett an die Fabrikklasse delegieren und geraten somit in keine Abhängigkeit.

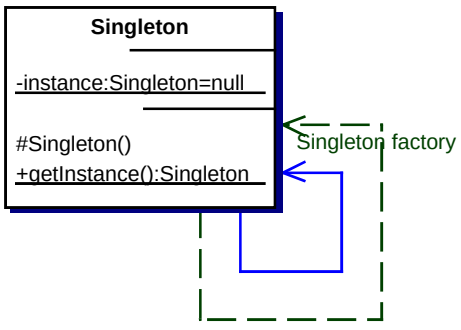

Abbildung 25 *- Klassendiagramm „Singleton“*

Die Abbildung zeigt das Klassendiagramm des nur aus einer Klasse bestehenden Musters *Singleton*. Entscheidend ist, dass der Konstruktor `Singleton()` als `protected` deklariert ist, so dass von außen keine Instanzen dieser Klasse erstellt werden können. Stattdessen ist der einzige öffentliche Zugriffspunkt über `getInstance()` gegeben. Diese Methode liefert (falls die Klasse schon instanziiert ist) eine Referenz auf die (einzige) vorhandene Instanz bzw. (die Klasse ist noch nicht instanziiert) erstellt über den Konstruktor `Singleton()` die (einzige) Instanz und liefert die Referenz.

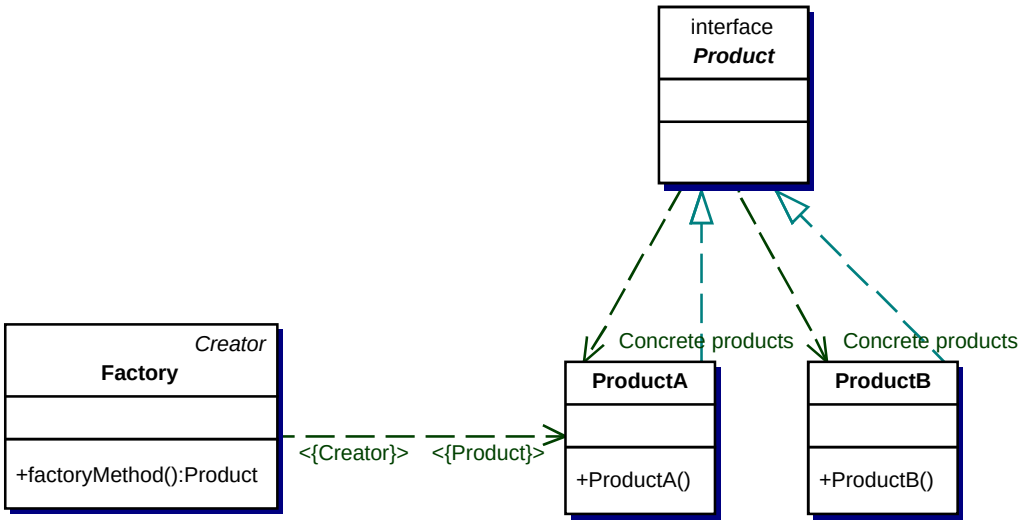

Abbildung 26 *- Klassendiagramm „Fabrikmethode“*

Die **Abbildung 26** zeigt die formale Klassenstruktur des Entwurfsmusters *Fabrikmethode*. Die Klasse `Factory` ist dabei mittels der Erzeugermethode `factoryMethod()` und anhand eines Diskriminators in der Lage, ein (durch den Diskriminator) bestimmtes konkretes Objekt vom Typ `Product` zu instanziieren und zurückzuliefern (in diesem Fall entweder die Instanz von `ProductA` oder von `ProductB`). Jede Klasse, die ein bestimmtes Objekt vom Typ `Produkt` benötigt, würde die Instanziierung dieser Klasse an die `Factory` delegieren, so dass sie selbst nichts von der Instanziierungsentscheidung wissen muss, sondern lediglich die Schnittstelle `Product` kennt. Selbst wenn sie Kenntnis über das konkrete `Produkt` besitzt, sollte sie trotzdem nur die `Produkt`-Schnittstelle nutzen und ihr Wissen lediglich in Form eines Diskriminators an die Fabrikmethode der `Factory`-Instanz weitergeben. Der Einsatz dieses Musters entkoppelt also verschiedene Bereiche von Anwendungen, indem spezielles Wissen in bestimmten Klassen zusammengefasst werden, was wiederum den Vorteil hat, das sich Änderungen bzw. Erweiterungen auch nur auf diese Klassen auswirken.

Abbildung 27 auf der folgenden Seite zeigt einen Ausschnitt aus dem Package „`temporalpersistence`“ der Gesamtanwendung, in dem die beiden beschriebenen

Muster enthalten sind. Die Klasse `TemporalStorageFactory` ist dabei sowohl *Singleton* als auch *Fabrik* und kann über die Methode `getInstance()` erreicht werden und über die Methode `createTemporalStorage()` in Abhängigkeit von Konfigurationsparametern in der Konfigurationsdatei „`web.xml`“ Instanzen vom Typ `TemporalStorage` erstellen. Bei diesen Instanzen handelt es sich um Klassen, die die Schnittstelle `TemporalStorage` entweder dateibasiert (`FileTemporalStorage`) oder datenbankbasiert (`SybaseDBTemporalStorage`) implementieren.

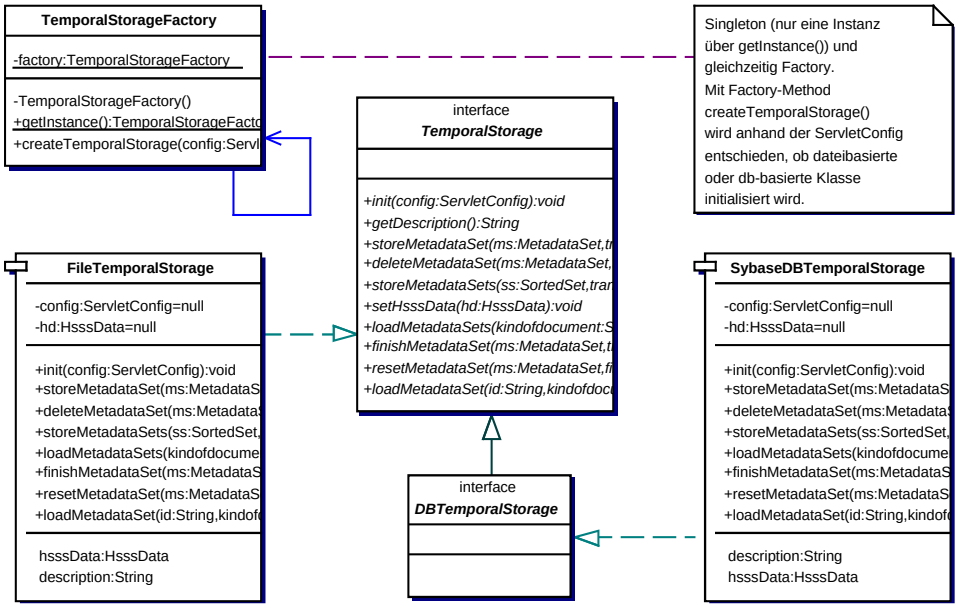

Abbildung 27 *- HSSS Package “temporalpersistence“ (Ausschnitt)*

Der Vorteil liegt in der Entkopplung und in dem Verstecken von Informationen (*information hiding*) in ganz bestimmten Klassen. Im betrachteten Beispiel bestand die einzige Instanziierungsmöglichkeit der Schnittstelle `DBTemporalStorage` in der Klasse `SybaseDBtemporalStorage` für eine Sybase-Datenbank. Würde sich die Datenbankanbindung in Zukunft ändern oder erweitern (durch zum Bsp. Oracle), dann müssten nur zwei Änderungen durchgeführt werden. Erstens wird mit der Klasse `OracleDBTemporalStorage` eine entsprechende Implementierung der Schnittstelle `DBTemporalStorage` (und damit auch `TemporalStorage`) geschaffen, und zweitens brauchen lediglich wenige Zeilen Quellcode innerhalb von `TemporalStorageFactory` geändert bzw. hinzugefügt werden. Wäre die Instanziierung der verschiedenen Klassen vom Typ `TemporalStorage` nicht durch eine *Fabrik* realisiert worden, sondern von jeder eine solche Instanz benötigenden Klasse selbst (z. Bsp. 50), dann hätte die Änderung auch bei **jeder** dieser Klassen (z. Bsp. 50) durchgeführt werden müssen.

3.3.3. Architekturdetails auf Client-Ebene

Die Architekturdetails auf Client-Ebene sind zum Teil recht abhängig von der jeweiligen Teilanwendung, weshalb solche Details dann auch in den entsprechenden Kapiteln erörtert werden sollen. Außerdem wurden eine Reihe von Aussagen bereits von der Belegarbeit getroffen [Schb00]. Deshalb sollen an dieser Stelle lediglich eine Einordnung in die Gesamtarchitektur sowie ein kurzer Ausblick auf technologische Alternativen erfolgen.

3.3.3.1. Einordnung in die Gesamtarchitektur

Die Architekturdetails auf der Client-Ebene sind abhängig von der gewählten Architektur des Gesamtmodells. Für Webanwendungen gibt es insbesondere zwei benannte Architekturansätze (siehe [AAB00], [PLC99]), die man verfolgen könnte. Der erste Architekturansatz, der als „*page centric approach*“[AAB00] bzw. „*Web Application Model 1*“ bekannt ist, ist in **Abbildung 28** zu sehen:

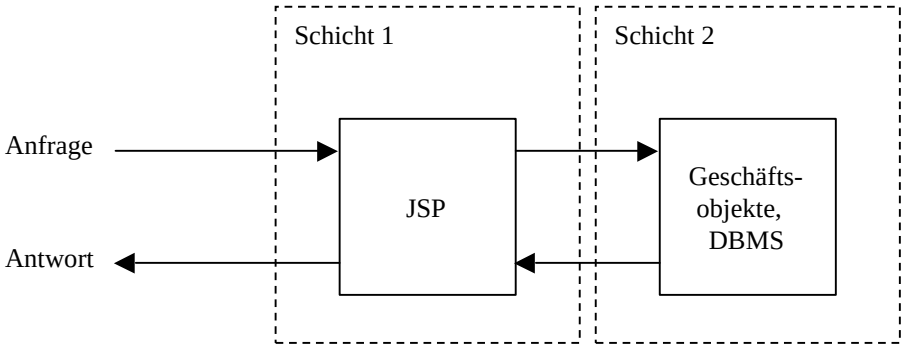

Abbildung 28 *- Architektur „page centric approach“ (2-Tier)*

Dieses Modell verfolgt einen Zwei-Schichten-Ansatz, bei dem die Java ServerPages in Schicht 1 sowohl die Benutzerinteraktionen als auch große Teile der Geschäftsfunktionalität (so auch direkte Zugriffe auf Daten) übernehmen. Es eignet sich eher für kleinere Webanwendungen und hat den spezifischen Nachteil, dass in Schicht 1 Layout (Repräsentation), Anwendungslogik und Datenstrukturen relativ stark vermischt werden. **Abildung 29** zeigt auf der folgenden Seite einen alternativen Ansatz, der das *Model-View-Controller*-Paradigma auf der Ebene von Webanwendungen relativ gut realisiert. Er wird in [AAB00] auch als „*dispatcher approach*“ bezeichnet, weil

das Servlet in Schicht 2 als *Controller* (oder *Dispatcher*) zwischen den verschiedenen JSP's der Schicht 1 (*View*) und den Geschäftsobjekten bzw. den Daten (*Model*) fungiert. Für die Java ServerPages in Schicht 1 hat das den Vorteil, dass sie hauptsächlich nur für das Layout, also für die Darstellung bestimmter Sachverhalte verantwortlich sind. So werden die einzelnen Anwendungsschwerpunkte relativ sauber voneinander getrennt, weshalb dieser Architekturansatz auch für größere Webanwendungen prädestiniert ist. Aufgrund dieser Tatsache wurde dieser Ansatz bereits während des Belegs für die Realisierung der einzelnen Teilanwendungen (Webanwendungen) des HSSS gewählt und wird auch im Diplom beibehalten.

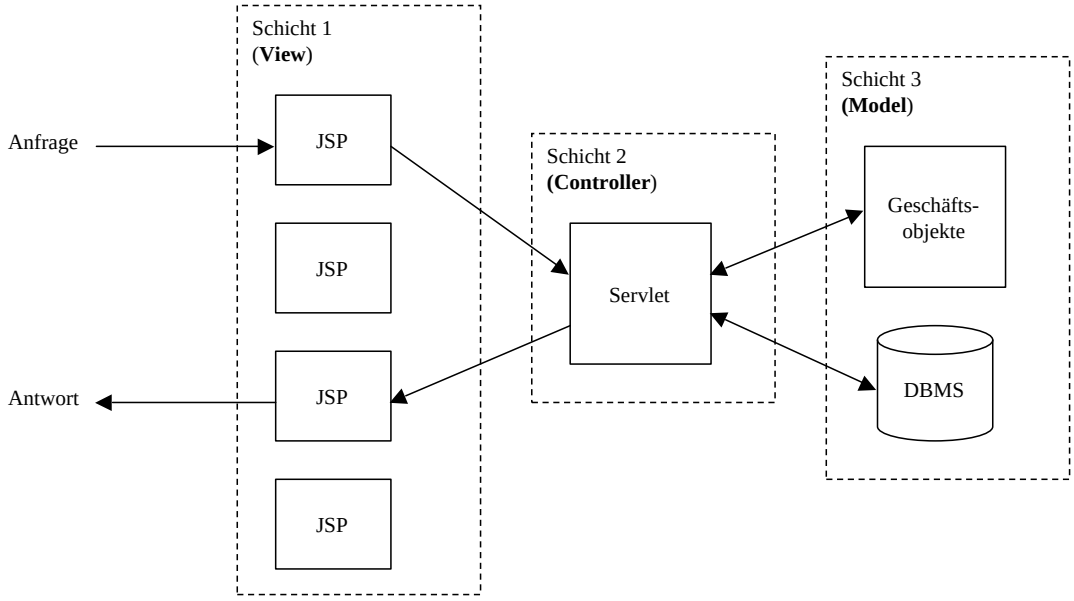

Abbildung 29 *- Architektur „dispatcher approach“ (MVC)*

Auf die Architekturdetails der Client-Ebene hat diese Entscheidung insofern Einfluss, als damit das Aufgabengebiet der Client-Seite klar abgegrenzt wird:

- Die Client-Seite der Webanwendungen des HSSS fungiert als *View* innerhalb des Architekturansatzes „dispatcher approach“
- Das bedeutet, dass die entsprechenden JSP's in der Regel nur layoutspezifische Aufgaben realisieren.
- Der Anteil von *Scriptlets* innerhalb der JSP's ist deshalb soweit wie möglich zu minimieren.

Bei Beachtung dieser Vorgaben ist auf der Client-Seite eine Implementation möglich, die nahezu alle Interna der Anwendung wie Flusskontrolle, Wissen über Datenstrukturen, Zugriff auf externe Ressourcen etc. anderen Bereichen überlässt und selbst nur die Darstellung von Information für den Anwender übernimmt. Dadurch eignet sich dieser Ansatz auch besonders, um in einem Team mit unterschiedlichen fachlichen Kompetenzen parallel zu entwickeln. So können Details der Anwendung von Softwarearchitekten etwa in Schicht 2 oder 3 realisiert werden, die dann nur noch entsprechende Schnittstellen kommunizieren bzw. dokumentieren müssen. Mit Hilfe dieser Schnittstellen kann dann ein Web-Designer passende Webseiten entwickeln, ohne alle Einzelheiten der Anwendung kennen oder entsprechende Programmiersprachen beherrschen zu müssen.

Allerdings verbleiben in den JSP's Quellcode-Teile (vor allem sogenannte *Scriptlets*), die Java-spezifisch sind und zur dynamischen Generierung von HTML-Code gebraucht werden. Diese Java-Anteile im Quellcode lassen sich auch dann nicht vollständig vermeiden, wenn man sich lediglich auf Layout-Funktionalitäten beschränkt und wirken zum Beispiel für einen Web-Designer innerhalb des ansonsten HTML-basierten Quellcodes entsprechend störend. Dieses Problem lässt sich durch die Verwendung von *TagLibs* (Tag-Bibliotheken) umgehen, welche es erlauben, für bestimmte Java-Quellcode-Teile Tags zu definieren, die sich dann wesentlich besser und übersichtlicher in den restlichen Quellcode einpassen[AAB00]. **Listing 7** zeigt den Unterschied formal an einem sehr simplen Beispiel, welches in der Praxis den Einsatz von *TagLibs* selbstverständlich noch nicht rechtfertigen würde.

```
1  ohne TagLib:
2   ...
3  <body>
4  <b>Es ist genau: </b>
5  <% Date d = new java.util.Date(); %>
6  <%= d.toString() %>
7  </body>
8   ...
9  mit TagLib:
10  ...
11 <%@ taglib uri="\myTagLib" prefix="myTags" %>
12  ...
13 <body>
14 <b>Es ist genau: </b>
15 <myTags:CurrntDate/>
16 </body>
17  ...
```

***Listing 7** : Einfaches Beispiel für den Einsatz von TagLibs*

Trotzdem lässt sich an dem Beispiel bereits gut erkennen, dass diese Technologie - richtig angewendet - zu einer weiteren Entkopplung von Präsentation und Anwendungslogik führt und damit auch zu einer noch besseren Trennung der verschiedenen

Rollen innerhalb des Entwicklungsprozesses von Webanwendungen[Roe00]. Die Definition solcher Bibliotheken[47] ist allerdings relativ aufwendig und verlangt eine gewisse Erfahrung, um möglichst flexibel einsetzbare Tags zu generieren. Deshalb wird diese Technologie im Rahmen der Diplomarbeit auch keine Anwendung finden, bleibt aber für die zukünftige Entwicklung des HSSS eine interessante Option (siehe dazu auch Abschnitt 8.2).

3.3.3.2. Technologische Alternativen

Ist die Entscheidung für eine Webanwendung gefallen, gibt es auf der Architekturebene clientseitig kaum weitere Alternativen neben den im vorangegangenen Abschnitt vorgestellten[48]. Neben der Realisierung der Anwendungsteile als Webanwendungen wären natürlich auch konventionelle, clientseitige Anwendungen möglich. Hier bestehen allerdinge einige schwere Nachteile, die noch einmal kurz zusammengefasst werden sollen:

- Eine solche Anwendung ist wesentlich stärker von den technischen Gegebenheiten auf der Client-Seite abhängig.
- Die Anwendung muss erst auf dem entsprechenden Rechner installiert werden (neben aller Software, die sie möglicherweise zur Ausführung benötigt). Dieser Installationsvorgang ist in der Regel nicht trivial.
- Aktualisierungen sind (bisher) automatisch kaum möglich und daher extrem aufwendig zu realisieren.
- Verwendet man keine plattformunabhängige Technologie, muss man auch noch für jede Plattform eine Portierung bereitstellen.

Selbstverständlich gibt es bei clientseitigen Anwendungen auch eine Reihe von Vorteilen:

- Die Nutzerschnittstelle ist in der Regel wesentlich reichhaltiger und erlaubt direktere und schnellere (auch zuverlässigere) Reaktionen auf die Nutzereingaben (da sie nicht von der Geschwindigkeit der Internetanbindung abhängig ist).

[47] geschieht über die Auslagerung der sich hinter den Tags befindlichen Logik in seperate Java-Klassen und XML-basierte Beschreibungsdateien

[48] Vielleicht mit Ausnahme von bestimmten Frameworks, die die Umsetzung des MVC-Paradigmas noch erleichtern sollen, wie beispielsweise Struts [L-ST].

- Die Anwendung ist für den Nutzer wie gewohnt zu erreichen und verlangt nicht die Existenz eines Browsers. Gerade für unerfahrene Anwender ist das ein wichtiger Punkt.
- Die Anwendung ist auch verfügbar, wenn der Klient „offline“ ist.

Diese Vorteile wiegen um so schwerer, je komplexer die betrachtete Nutzerschnittstelle ist und je kritischer Eigenschaften wie zuverlässige Verfügbarkeit und Antwortzeitverhalten sind. Seit einiger Zeit gibt es nun im Java-Bereich eine Technologie, die es laut *SUN* ermöglicht, die Vorteile der beiden Ansätze miteinander zu kombinieren. Die Technologie heißt *„Java WebStart“*[L-JW] und basiert auf dem *Java Network Launcher Protocol* (JNLP), welches ein Protokoll für ein webzentriertes Anwendungsmodell im Bereich von Java-Technologie realisiert[Hol00]. Eine Anwendung ist demnach eine Ressource, die sich an einem beliebigen Ort im Netz befindet. Der Anwender entscheidet, ob das Programm geladen und ausgeführt werden soll und welche Rechte es dabei hat. *WebStart* ist lediglich eine Referenzimplementation eines JNLP-Clients. Folgende Schritte werden vom Protokoll und den beteiligten Komponenten realisiert (stark vereinfacht):

- Der JNLP-Client (also zum Bsp. *WebStart*) holt die benötigten Programmteile vom Webserver des Herstellers und speichert diese im lokalen Cache. Eine sogenannte JNLP-Datei enthält dabei (XML-basiert) alle Deployment-Informationen des entsprechenden Programms, so dass für den Anwender kein Installationsprozess notwendig ist.
- Der JNLP-Client kann die Ressourcen im lokalen Cache mit denen auf dem Server vergleichen und somit entscheiden, ob neuere Programmteile nachgeladen werden müssen. Eine solche Aktualisierung ist für den Anwender transparent.
- Dieser Mechanismus erlaubt eine sehr feine Teilung von Anwendungen in einzelne Ressourcenteile. So ist es möglich, die Aktualisierung inkrementell vorzunehmen und zum Beispiel nur einzelne Klassen nachzuladen, was zu einer erheblichen Reduzierung der Downloadzeit führt.
- Das JNLP-Protokoll ermöglicht ebenfalls eine inkrementelle Installation, so dass Programmteile, die vorerst nicht benötigt werden, auf dem Server verbleiben. Sie können im später in Form einer inkrementellen Aktualisierung nachgeladen werden.

Die Java *WebStart*-Technologie bzw. das JNLP-Protokoll ermöglicht dabei für solche Anwendungen Sicherheitsmerkmale, wie sie von der Java 2 Plattform bzw. von

Applets bekannt sind, also insbesondere den Ablauf in geschützten Bereichen[49] (Sandboxing). Damit könnte sich JNLP und die *WebStart*-Technologie zu einem wichtigen Baustein in der Java 2 Plattform entwickeln, der die Brücke zwischen laufendem Betriebssystem und Web-zentrierten Java-Anwendungen schlägt. Allerdings besteht die Voraussetzung, dass sich ein solcher JNLP-Client (z. Bsp. *WebStart*) lokal auf dem Rechner befindet. Ideal wäre demnach die Integration eines solchen Clients in das Betriebssystem.

Für die Entwicklung der HSSS-Teilanwendungen im Rahmen des Diploms kommt diese Technologie jedoch nicht in Frage, da sich die zu realisierenden Benutzerschnittstellen nach Meinung des Autors auch sehr gut (und teilweise wesentlich effizienter) webbasiert gestalten lassen. Außerdem steckt die hier betrachtete Technologie teilweise noch „in den Kinderschuhen“ und muss ihre Wirksamkeit in der Praxis erst unter Beweis stellen. Als Alternative zu den webbasierten Schnittstellen wären zusätzlich vorhandene clientseitige Anwendungen in einer weiteren Ausbaustufe des HSSS aber sicherlich interessant, wobei dies für die einzelnen Teilanwendungen in unterschiedlichem Maße gilt[50] (siehe dazu auch Abschnitt 8.2 ab Seite 222).

3.3.4. Architekturdetails auf Datenbank-Ebene

In diesem Abschnitt sollen lediglich einige sehr allgemeine Aussagen zur Datenbankarchitektur getroffen werden, da sich die noch folgenden Kapitel zu dem einzelnen Teilanwendungen mit speziellen Fragen wie z. Bsp. dem Schemaentwurf beschäftigen werden. Zunächst werden die grundlegendsten Anforderungen des Gesamtsystems an eine Datenbank bzw. an ein Datenmodell aufgelistet, danach erfolgt eine formale Datenbank-Evaluation anhand dieser Kriterien.

[49] Dabei ist die Definition verschiedener Sicherheitsstufen möglich, z. Bsp. *„untrusted-environment-permissions“*, *„j2ee-application-client-permissions“* oder *„all-permissions“*

[50] So wäre eine solche Realisierung insbesondere für die Teilanwendung *AdminTools* interessant, weil es sich hier um die mit Abstand komplexeste Schnittstelle handelt, die außerdem in den Punkten Performance und Stabilität unter Umständen stark dazugewinnen könnte. Auch die alternative Realisierung des *MetadatenGenerators* würde auf diesem Wege Sinn machen, das gilt jedoch kaum für die dritte Teilanwendung *AccessTools*.

3.3.4.1. Schema- und Datenbankanforderungen

Dadurch, dass die Datenbank laut Aufgabenstellung lediglich die Metadatensätze zu den einzelnen Hochschulschriften und nicht notwendigerweise die Dokumente selbst enthalten soll, sind die Anforderungen an das noch zu spezifizierende Schema und damit auch an die Datenbank sehr moderat. Insbesondere liegen die Wertebereiche der Informationen, die in den Metadatensätzen enthalten sind, im Bereich konventioneller Datentypen, wie sie in nahezu allen Datenbanken existieren[51].

Ein Kriterium, welches die Auswahl an Datenbankverwaltungssytemen möglicherweise ein wenig mehr einschränkt, ist die zu fordernde Unterstützung von Transaktionen. Schon in diesem frühen Stadium des Entwurfs ist absehbar, dass einige Datenbankoperationen mit Sicherheit innerhalb einer Transaktion ablaufen müssen, so zum Beispiel das Löschen des temporären Metadatensatzes und das Einstellen desselben in die endgültig persistente Datenbasis des Gesamtsystems.

Ein nächster Punkt, der den Entwurf des Datenbank-Schemas, aber auch die Evaluation der Datenbank beeinflussen könnte, kann aus den erwarteten Anfragen abgeleitet werden, insbesondere, was deren vermutete Komplexität bzw. Ausrichtung betrifft. Für die Teilanwendungen *MetadatenGenerator* und *AdminTools* besteht die Mehrzahl der Anfragen darin, die temporären Metadatensätze aus der Datenbank auszulesen und (geändert) wieder abzuspeichern. Am Ende der Bearbeitung durch die *AdminTools* wird der Metadatensatz endgültig persistent abgelegt, aufgrund seiner geordneten, hierarchischen Struktur ist eine zumindest ähnliche Struktur des Schemas zu erwarten. Alle weiteren Anfragen in Bezug auf einen solchen Metadatensatz sind lesende Anfragen (vor allem durch die Teilanwendung *„AccessTools“*), welche die Metadateninformationen z. Bsp. für einen Suchvorgang oder eine Navigation liefern sollen. Zusammenfassend lässt sich also sagen, dass insbesondere auf die endgültig persistent vorliegenden Metadatensätze mehrheitlich lesend zugegriffen werden wird, wobei diese Zugriffe relativ zeitkritisch und daher durch das Schema bzw. durch die Datenbank so gut wie möglich zu unterstützen sind.

Im Hinblick auf die technologische Ausrichtung des Gesamtsystems lassen sich noch weitere Kriterien aufzählen, die wünschenswert, jedoch nicht existenziell sind. So wäre eine Integration von Java, z. Bsp. durch die Unterstützung als Datentyp oder als mögliche Sprache für gespeicherte Prozeduren oder Trigger, sehr vorteilhaft. Ebenso wünschenswert wäre eine XML-Unterstützung insbesondere für den Zugriff auf XML-

[51] etwa Integer, Varchar(n), Date etc.

basierte Strukturen innerhalb der Datenbank[52]. Zusätzliche Dienste, wie z. Bsp. die Integration einer Volltext-Suchmaschine, wären ebenfalls ein hervorragendes Argument für ein solches Datenbankverwaltungssystem.

3.3.4.2. Datenbank-Evaluation

Die Datenbankevaluation soll an dieser Stelle in erster Linie allgemein, d.h. nicht bezogen auf bestimmte Hersteller oder Produkte, durchgeführt werden. Dazu werden die zur Zeit üblichen Datenbankverwaltungssysteme grob in die folgenden Kategorien unterteilt:

- Objektorientierte Datenbankverwaltungssysteme (ODBMS)
- XML-Datenbankverwaltungssysteme
- Relationale Datenbankverwaltungssysteme (RDBMS[53])
- Open Source - Datenbankverwaltungssysteme[54]

Eine Objektorientierte Datenbank hätte den Vorteil, Objektstrukturen direkt persistent ablegen zu können. Das wäre besonders dann von Vorteil, wenn das zu erwartende Schema äußerst komplex ist, weil das Datenmodell auch in diesem Fall den schnellen Zugriff auf die Gesamtstruktur erlaubt. Allerdings ist eine solche Komplexität für das Datenbank-Schema des HSSS nicht zu erwarten. Ein Nachteil solcher Datenbanksysteme ist, dass sie die Zuverlässgkeit und Performance von relationalen Datenbanksystemen (ein nicht extrem partitioniertes Schema vorausgesetzt) in der Regel nicht erreichen. Da das voraussichtliche Schema des HSSS keinen überdimensionalen Komplexitätsgrad erwarten lässt und der Fokus eindeutig auf Stabilität und Geschwindigkeit liegt, spielen objektorientierte Datenbankverwaltungssysteme im weiteren Verlauf der Evaluation keine Rolle.

[52] z. Bsp. durch die Möglichkeit, direkt auf Teilstrukturen zugreifen zu können, durch ein automatisches Umsetzen von XML-Strukturen in das jeweilige interne Datenmodell etc.

[53] wobei zu dieser Kategorie auch die Postrelationalen bzw. Objektrelationalen Datenbankverwaltungssysteme (ORDBMS) gezählt werden sollen

[54] in diesem Sinne natürlich keine technische Einteilung, diese Datenbanken können also jeweils einer der Kategorien ODBMS, XML-DBMS oder RDBMS angehören

Ein XML-basiertes Datenbanksystem hätte für den HSSS den großen Vorteil, dass die vielfach vorhandenen XML-Strukturen sehr elegant (nämlich direkt) durch die Datenbank behandelt werden könnten. Sie bräuchten nicht durch die Anwendung geparst zu werden, sondern könnten direkt in der Datenbank geändert werden. Allerdings gibt es nicht sehr viele echte XML-Datenbanken (mit wirklichem XML-Datenmodell) und die existierenden sind technisch bei weitem nicht so ausgereift wie etwa objektorientierte oder gar relationale Datenbanksysteme. Eine direkte Behandlung der XML-Strukturen durch die Datenbanken wäre zwar schön, allerdings ist das aufgrund der Flexibilität von XML nicht zwingend erforderlich, so dass XML-basierte Datenbanken aufgrund ihrer Stabilitäts- bzw. Performancenachteile für den Einsatz ebenfalls nicht in Frage kommen.

Relationale Datenbanksysteme sind immer noch die am meisten eingesetzten Datenbanken, insbesondere dann, wenn es um Stabilität und Performance auch bei größten Datenmengen geht und ein nicht zu stark partitioniertes Schema zugrunde liegt (was in der Regel der Fall ist). Da diese Systeme vergleichsweise lang im Einsatz sind, wurden sie im Verlauf der Jahre entsprechend ausgiebig optimiert und weiterentwickelt. Zudem integrieren immer mehr relationale Systeme weitergehende Funktionalitäten, die teilweise stark in den Bereich objektorientierter bzw. XML-basierter Datenbanksysteme hineinreichen, so dass auch von diesem Blickwinkel aus betrachtet die wenigen Nachteile relationaler Datenbanken kaum ins Gewicht fallen. Als Datenbankverwaltungssystem innerhalb des HSSS soll daher ein relationales Datenbanksystem eingesetzt werden, was in der folgenden Aufzählung von Vorteilen noch einmal detaillierter begründet wird:

- RDBMS sind in den Punkten Zuverlässigkeit, Stabilität und Performance nach wie vor unübertroffen.
- RDBMS sind trotz des großen Interesses an objektorientierten Datenbanken Mitte der 90-er Jahre die Systeme, die im Produktionsbetrieb am häufigsten eingesetzt werden.
- Moderne RDBMS verfügen über zusätzliche Funktionalitäten, welche einige der Schwachpunkte des Relationenmodells zumindest entschärfen (Postrelationale bzw. Objektrelationale Systeme).
- Die Metadatensätze des HSSS als die einzigen im Datenbank-Schema abzubildenden Strukturen eignen sich hervorragend für das Relationenmodell.
- RDBMS bieten ausgereifte und äußerst gut erforschte Realisierungen für Transaktionen.

- Im Vergleich zu anderen Datenbank-Technologien besteht bei RDBMS eine breite Auswahl an entsprechenden Systemen.

Die Wahl fällt also generell auf ein relationales Datenbankverwaltungssystem. Dabei ist im Folgenden abzuwägen, ob ein kommerzielles Produkt oder eine Open Source-Alternative eingesetzt werden soll. Die Entscheidung fällt in diesem Fall zugunsten eines kommerziellen Produktes aus, weil diese Systeme im Vergleich mit Open Source-Datenbanken in den meisten Fällen wesentlich umfangreicher und leistungsfähiger sind, und zudem aufgrund existierender Landeslizenzen die Möglichkeit bestand, zwischen zwei kommerziellen Datenbanksystemen sehr preiswert wählen zu können. Bei diesen Datenbanken handelt es sich um die Produkte von *Oracle* (*Oracle8i Database Server*[L-OR]) und *Sybase* (*Adaptive Server Enterprise* (ASE) [L-SY]).

Obwohl es sich bei dem Produkt von *Oracle* um das aktuellere, umfangreichere und auch weiter verbreitete handelt, fiel die Entscheidung zugunsten des Datenbankservers von *Sybase* aus. Die ausschlaggebenden Gründe dafür waren, dass bei der SLUB bereits Erfahrungen in der Administration von Sybase-Datenbanksystemen vorhanden waren und dass die Lizenz für dieses Datenbankverwaltungssystem im Rahmen der Landeslizenz noch einmal wesentlich günstiger war. Zudem wären die zusätzlichen Fuktionalitäten des *Oracle*-Datenbankservers für die Anwendung nur begrenzt von Vorteil gewesen. Allerdings handelt es sich bei der zu diesem Zeitpunkten aktuellen Version 12.0 des *Adaptive Server Enterprise* von *Sybase* um eine Version, die sich bereits geraume Zeit am Markt befindet und in näherer Zukunft durch die nachfolgende Version 12.5 abgelöst wird. Diese Version bietet erhebliche Weiterentwicklungen insbesondere in den Bereichen XML- und Java-Unterstützung, so dass eine Aktualisierung zur gegebenen Zeit durchaus Sinn machen würde (siehe dazu auch Abschnitt 8.2 ab Seite 222).

3.3.4.3. Design der Schnittstelle

Die Schnittstelle zur Datenbankanbindung soll sich natürlich möglichst nahtlos in die Gesamtanwendung und insbesondere in die Persistenzschnittstelle (die unabhängig davon ist, ob die hier betrachtete Datenbank oder eine dateibasierte Lösung verwendet wird) einfügen und dabei trotzdem flexibel und lose gekoppelt bleiben. **Abbildung 30** zeigt die wichtigsten Klassen des Package „`db`“, welches sich innerhalb des „`utilities`“-Package befindet und den Zugriffspunkt auf Datenbanken für die Anwendung realisiert:

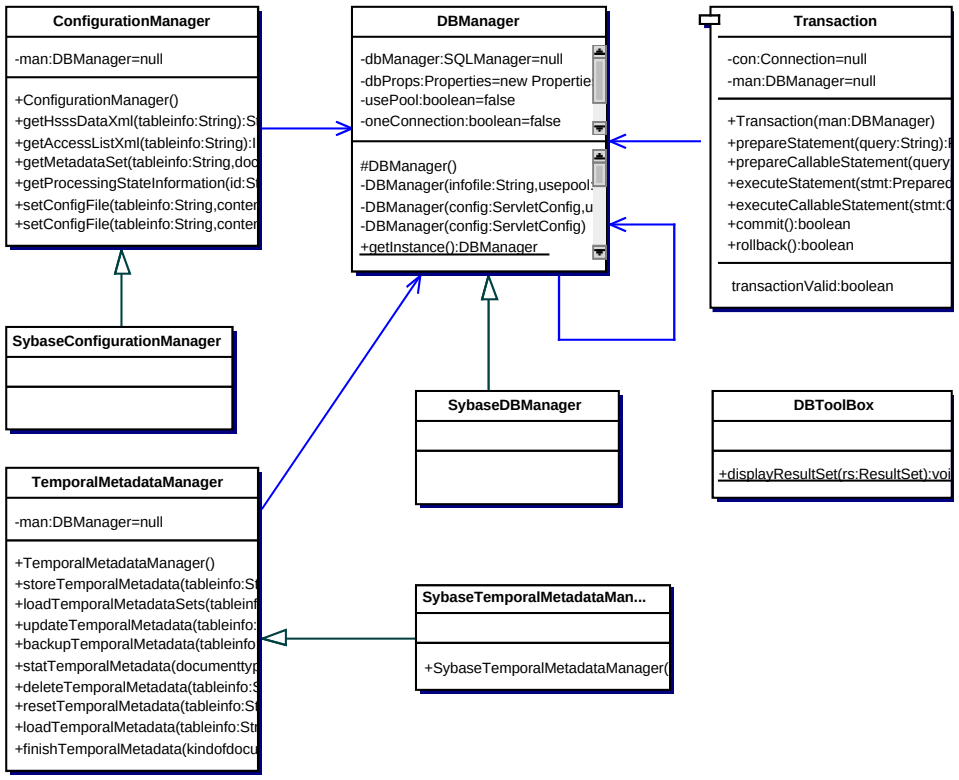

Abbildung 30 *- HSSS Package “db“*

Die Abbildung zeigt als wichtigste Instanz die Klasse `DBManager`, die als *Singleton* realisiert ist und den zentralen Zugriffspunkt der Anwendung auf Datenbanken darstellt. Der `DBManager` fungiert dabei als *Wrapper*, der Interna vor der Anwendung verbirgt und stattdessen die Interaktion über wenige, wohldefinierte Methoden erlaubt, so liefert die Klasse Datenbankverbindungen (`Connection`) und Transaktionsobjekte (`Transaction`) und ermöglicht weitere Datenbankmanipulationen. Von außen

können dabei über Konfigurationsstrukturen Teile des Verhaltens bestimmt werden[55]. Die Klasse `DBManager` kann dabei für bestimmte Datenbanken von weiteren Klassen spezialisiert werden, so wie in der Abbildung die Klasse `SybaseDBManager`. Die Klasse `Transaction` stellt ein Objekt zur Verfügung, welches für die Anwendung eine Transaktion kapselt. Alle Befehle, die über dieses Objekt (besser: über dessen `Connection`-Object) an die Datenbank abgesetzt werden, gehorchen dem ACID-Prinzip[56] und werden somit gemeinsam und vollständig oder aber gar nicht ausgeführt. Dazu bietet die Klasse typische Methoden wie `commit()` bzw. `rollback()`. Daneben existieren zwei Klassen für die zwei typischen Bereiche, in denen die Anwendung externe Daten liest bzw. schreibt: der Umgang mit Konfigurationsstrukturen bzw. mit Metadatensätzen. Für den ersten Fall liefert die Klasse `ConfigurationManager` die notwendige Funktionalität, um auf datenbankbasierte Konfigurationsstrukturen zugreifen zu können. Die Klasse `TemporalMetadataManager` erlaubt die Behandlung von temporären Metadatensätzen (Speichern, Lesen, Löschen, Ändern, Rücksetzen etc.) durch die Anwendung. Beide Klassen können jeweils für bestimmte Datenbanken spezialisiert werden.

[55] So z. Bsp. ob *Connection Pooling* verwendet werden soll und mit welchen Parametern (Poolgrößen, Cachegrößen etc.)

[56] Das ACID-Prinzip von Transaktionen: **A**tomicity, **C**onsistency, **I**solation, **D**urability

4. Teilanwendung 1 : MetadatenGenerator

Inhalt des Kapitels:

4.1. Objektorientierte Analyse und Design 122
4.2. Realisierung 129
4.3. Persistenzrealisierung 134
4.4. Zusätzliche Funktionalitäten 136
4.4.1. Evaluation 136
4.4.2. Status-Verfolgung durch den Autor 137
4.4.3. MetadatenGenerator Light - eine Alternativ-Version 139
4.5. Einschätzung 142

4.1. Objektorientierte Analyse und Design

In diesem Abschnitt sollen spezielle Aspekte der objektorientierten Analyse und des Designs, die sich auf die Teilanwendung *MetadatenGenerator* beziehen, näher erläutert werden. Obwohl diese Teilanwendung im Rahmen des Großen Belegs bereits umfassend und zufriedenstellend implementiert werden konnte, ist sie im Verlauf der Diplomarbeit nahezu vollständig neu implementiert worden (freilich unter Nutzung und Übernahme der positiven Eigenschaften sowie des Layouts aus dem Beleg). Die Gründe ergaben sich vor allem aus den in der Aufgabenstellung und in den Architekturvisionen (siehe 3.2.1 ab Seite 80) genannten Anforderungen. **Abbildung 31** zeigt das Klassendiagramm des Packages `metadatagenerator`. Strukturell gleicht es den Packages der anderen Teilanwendungen, so dass es an dieser Stelle exemplarisch ebenfalls für die beiden noch verbleibenden Teilanwendungen *AdminTools* und *AccessTools* (und deren Packages `admintools` und `accesstools`) behandelt wird.

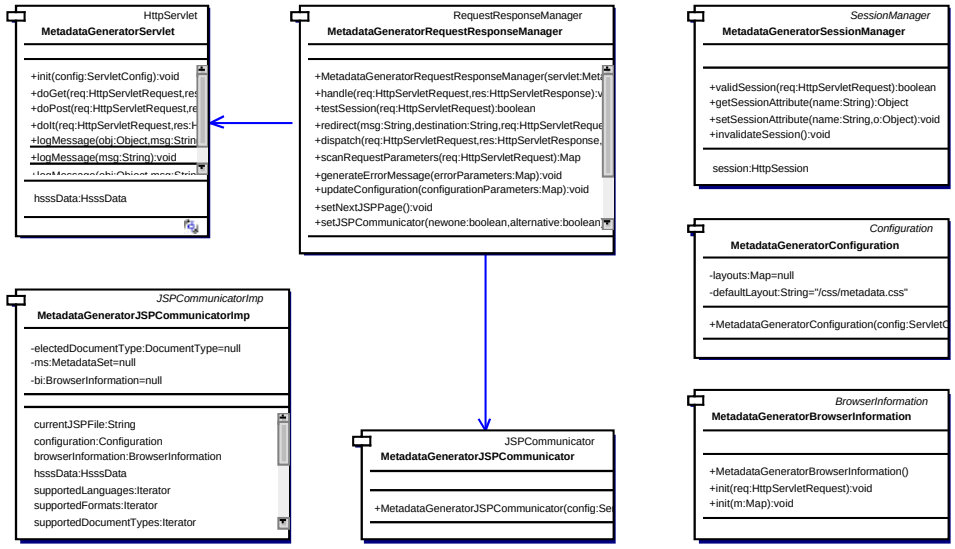

Abbildung 31 *- HSSS Package "metadatagenerator"*

Öffentlicher Zugriffspunkt auf die Webanwendung *MetadatenGenerator* ist die Klasse (bzw. das Servlet) `MetadataGeneratorServlet`, welche von `HttpServlet` erbt. Wie in Abschnitt 3.3.1.2 bereits kurz angedeutet, erhält das Servlet vom Servlet-Container Zugriff auf eine Reihe von Informationen (über `ServletConfig`), die zuvor in der Konfigurationsdatei „`web.xml`“ spezifiziert werden können. Aufgabe der Servlet-Klasse ist es, HTTP-Anfragen entgegenzunehmen, zu bearbeiten bzw. zu delegieren und entsprechende Antworten zurückzugeben. Dabei bedient sich die Servlet-Klasse der Methoden

`doGet()` (für HTTP-GET-Requests) bzw. `doPost()` (für HTTP-POST-Requests), die im Falle der `MetadataGeneratorServlet`-Klasse an eine einheitliche `doIt()`-Methode weitergeleitet werden. Von dieser Methode wird die Klasse `MetadataGenerator-RequestResponseManager` genutzt, um alle weiteren Aktivitäten im Zusammenhang mit den HTTP-Anfragen zu behandeln. Dessen Methode `handle()` verrichtet dabei die Hauptarbeit, indem die Parameter der HTTP-Requests geparst und in entsprechenden Datenstrukturen abgelegt werden und anschließend die Bearbeitung nach einem bestimmten Schema erfolgt. Die Klasse `MetadataGeneratorRequestResponseManager` benutzt ihrerseits die Klasse `MetadataGeneratorSessionManager`, um die jeweilige Anfrage auf eine gültige *Session* zu testen und diese gegebenenfalls zu erstellen. Ist die HTTP-Anfrage gültig (bzw. deren *Session*), so werden die Parameter der Anfrage ausgewertet und gegebenenfalls die Objekt-Repräsentation des Metadatensatzes aktualisiert. Parameter können aber auch Auswirkungen auf die Klassen `MetadataGeneratorConfiguration` und `MetadataGeneratorBrowserInformation` haben. Im ersten Fall handelt es sich um eine Klasse, die Informationen zur aktuellen Konfiguration[1] der Teilanwendung (in Abhängigkeit von der *Session*) enthält. Im zweiten Fall enthält die Klasse Informationen zum verwendeten Browser (in Abhängigkeit der *Session*). Dies ist notwendig, um die für die *View* verantwortlichen Java ServerPages entsprechend zu informieren, da in Abhängigkeit vom jeweils in der *Session* verwendeten Browser bei der Realisierung des Layouts teilweise unterschiedliche Aktionen notwendig sind. Nachdem die Anfrage korrekt ausgewertet und behandelt werden konnte, wird sie an den Aufrufer zurückgegeben. Dafür stehen der Klasse `MetadataGeneratorRequestResponseManager` die Methoden `redirect()` (führt `sendRedirect()` aus) und `dispatch()` (führt `forward()` aus) zur Verfügung, welche das entsprechende JSP ansprechen, das als nächstes die dynamische Generierung der Nutzerschnittstelle realisiert. Die Klasse `MetadataGeneratorJSPCommunicator`[2] wird dabei benutzt, um die *Session*-basierte Kommunikation zwischen Servlet und jeweiligem JSP (und umgekehrt) zu realisieren. Dazu wird der jeweiligen *Session* ein Objekt vom Typ `MetadataGeneratorJSPCommunicator` hinzugefügt, welches seinerseits Methoden zur Verfügung stellt, die von den jeweiligen JSP's genutzt werden, um an die entsprechenden Informationen zu gelangen. Zu diesen Informationen gehören im Fall der Teilanwendung *MetadatenGenerator* natürlich in erster Linie Informationen zu dem jeweiligen Metadatensatz, den der Anwender mit Hilfe dieser Anwendung Schritt für Schritt erstellt. Da dieser Vorgang (siehe Belegarbeit [Schb00]) nach wie vor ein iterativer Vorgang ist, entsteht in etwa die im folgenden Sequenzdiagramm in **Abbildung 32** auf der nächsten Seite gezeigte Aktionsfolge.

[1] Damit ist die vom Anwender zur Laufzeit veränderbare Konfiguration der Teilanwendung *MetadatenGenerator* gemeint. Hier sind Parameter wie JavaScript-Validierung, E-Mail-Benachrichtigung und verschiedene Layouts der Webanwendung einstellbar.

[2] Diese Klasse ist im Zusammenhang mit der Klasse `MetadataGeneratorJSPCommunicatorImp` nach dem Entwurfsmuster „Bridge“ zu betrachten.

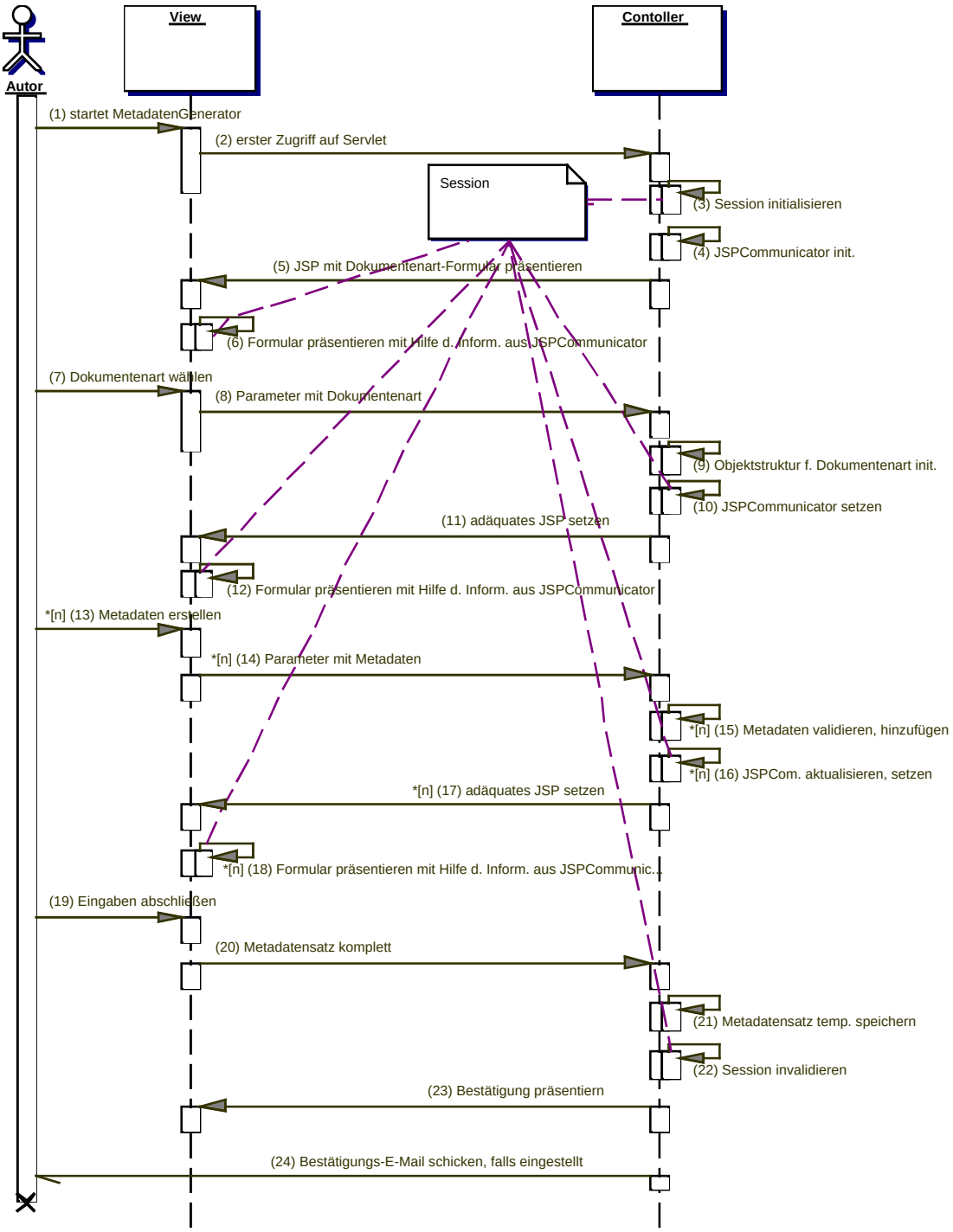
Autor
View
Contoller
Session
(1) startet MetadatenGenerator
(2) erster Zugriff auf Servlet
(3) Session initialisieren
(4) JSPCommunicator init.
(5) JSP mit Dokumentenart-Formular präsentieren
(6) Formular präsentieren mit Hilfe d. Inform. aus JSPCommunicator
(7) Dokumentenart wählen
(8) Parameter mit Dokumentenart
(9) Objektstruktur f. Dokumentenart init.
(10) JSPCommunicator setzen
(11) adäquates JSP setzen
(12) Formular präsentieren mit Hilfe d. Inform. aus JSPCommunicator
*[n] (13) Metadaten erstellen
*[n] (14) Parameter mit Metadaten
*[n] (15) Metadaten validieren, hinzufügen
*[n] (16) JSPCom. aktualisieren, setzen
*[n] (17) adäquates JSP setzen
*[n] (18) Formular präsentieren mit Hilfe d. Inform. aus JSPCommunic...
(19) Eingaben abschließen
(20) Metadatensatz komplett
(21) Metadatensatz temp. speichern
(22) Session invalidieren
(23) Bestätigung präsentiern
(24) Bestätigungs-E-Mail schicken, falls eingestellt

Abbildung 32 *- View-Controller-Aktivitäten des MetadatenGenerators*

Am Anfang beginnt der Autor die Aktionsfolge, indem er den *MetadatenGenerator* startet **(1)**. Seine Anfrage wird automatisch an den *Controller* weitergeleitet

(2), welcher erkennt, dass es sich um einen initialen Zugriff handelt. Der *Controller*[3] initialisiert daher ein *Session*-Objekt **(3)** und fügt diesem ein Objekt vom Typ `MetadataGeneratorJSPCommunicator` als Attribut hinzu **(4)**. Dieses Objekt enthält bereits Informationen, die sich zum Beispiel aus der Konfigurationsstruktur „`hsssdata.xml`“ ergeben. Anschließend übergibt der *Controller* die Kontrolle an das entsprechende JSP der *View* **(5)**, in diesem Fall an jenes, welches das Formular mit den unterstützten Dokumentenarten präsentieren kann. Dieses JSP erstellt dann die dynamische HTML-Benutzerschnittstelle unter Nutzung von Informationen des `MetadataGeneratorJSPCommunicator`-Objektes **(6)**.

Der Autor wählt nun die gewünschte Dokumentenart (z. Bsp. „Dissertation“) **(7)**, dieser Parameter wird an den *Controller* weitergeleitet **(8)**, welcher daraufhin eine entsprechende Objektstruktur initialisiert und die jeweilige Konfigurationsstruktur einliest (z. Bsp. „`MetadataSetDissertation.xml`“) **(9)**. Die neuen Informationen werden im `MetadataGeneratorJSPCommunicator` aktualisiert, welcher als Attribut der *Session* generell verfügbar ist **(10)**. Anschließend wird die Anfrage an ein adäquates JSP weitergeleitet **(11)**, welches wieder mit Hilfe des `MetadataGeneratorJSP-Communicator`-Objektes den nächsten Teil der Benutzeroberfläche aufbaut **(12)**. Die folgenden 6 Aktionen werden in dieser Reihenfolge mehrfach hintereinander ausgeführt[4], was im Sequenzdiagramm durch „***[n]**“ symbolisiert ist. Der Autor erstellt jeweils ein Teil des Metadatensatzes **(13)**, dieser Teil wird in Form von HTTP-Parametern an den *Controller* weitergeleitet **(14)**, der die Parameter validiert und der gegenwärtigen Metadatensatz-Repräsentation hinzufügt **(15)**. Danach wird wieder das `MetadataGeneratorJSPCommunicator`-Objekt entsprechend aktualisiert **(16)** und die Anfrage der jeweils passenden JSP übergeben **(17)**, die ein entsprechendes Formular dynamisch erstellt **(18)**. Nach einer bestimmten Anzahl von Iterationen schließt der Autor die Eingabe der Metadaten ab **(19)**. Der *Controller* erhält ein entsprechendes Signal in Form eines ausgezeichneten Parameters **(20)** und validiert daraufhin den gesamten Metadatensatz, um ihn anschließend temporär zu speichern **(21)**. Anschließend invalidiert er die *Session* **(22)** und veranlasst, dass dem Autor eine Bestätigung präsentiert wird **(23)**. Falls die E-Mail Adresse des Autors dem System bekannt ist und er diese Option aktiviert hat, erhält er außerdem eine Bestätigungsnachricht per E-Mail **(24)**, in der ihm nochmals die Bearbeitungsnummer[5] seines Metadatensatzes

[3] Unter *Controller* werden in diesem Zusammenhang das Servlet `MetadatageneratorServlet` und die oben beschriebenen weiteren Klassen des Packages `metadatagenerator` verstanden.

[4] Da sich der gesamte vom Autor zu erstellende Metadatensatz auf mehrere Formulare respektive JSP's verteilt.

[5] Über die Bearbeitungsnummer kann jederzeit der Bezug zu dem entsprechenden Metadatensatz hergestellt werden, da diese ihn eindeutig bestimmt, und das solange, wie der Metadatensatz im System verbleibt (theoretisch ewig). Desweiteren hat die Bearbeitungsnummer im Zusammenhang mit einer zusätzlichen Anwendung für den Autor eine besondere Funktion, die in Abschnitt 4.4.2

mitgeteilt wird. Die bisher nur vage mit „Objektstruktur“ beschriebene Repräsentation des Metadatensatzes soll im Folgenden anhand der **Abbildung 33** näher erläutert werden:

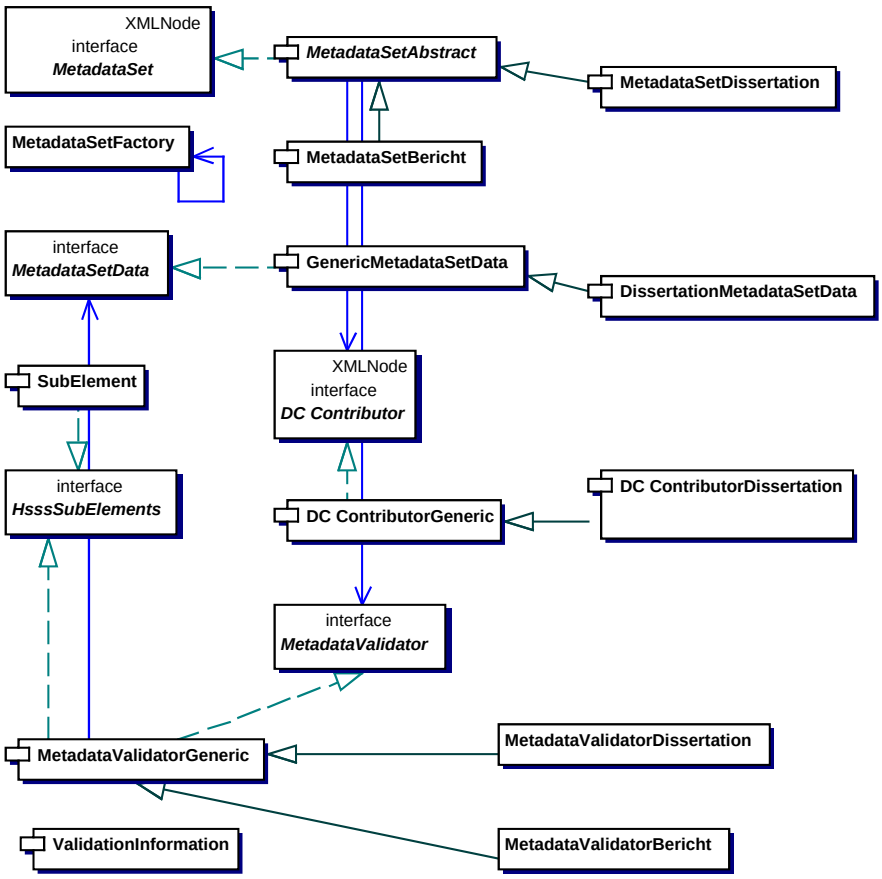

Abbildung 33 *- HSSS Package ”metadataset” (Ausschnitt)*

Die Abbildung zeigt einen Ausschnitt aus dem Klassendiagramm des Packages „`metadatset`“. Die allgemeinste Beschreibung eines Metadatensatzes existiert mit der Schnittstelle `MetadatSet`, welche die Grundfunktionalität der Metadatensätze aller Dokumentenarten bestimmt. Diese Schnittstelle wird zunächst von der Klasse `MetadataSetAbstract` implementiert, von welcher sich die Klassen `MetadataSetDissertation` und `MetadataSetBericht` für die beiden zur Zeit vom HSSS unterstützten Dokumentenarten „Dissertation“ und „Bericht“ ableiten. Welche dieser beiden Klassen instanziiert wird, entscheidet die Klasse `MetadatSetFactory` anhand der Wahl des Autors im *MetadatenGenerator*. Diese Klasse realisiert zugleich die Entwurfsmuster *„Singleton“* und *„Fabrikmethode“*. Jede Instanz vom Typ `MetadataSet`

ab Seite 137 näher beschrieben wird.

enthält eine Referenz auf ein Objekt vom Typ `MetadataValidator` bzw. dessen Spezialisierungen `MetadataValidatorGeneric`, `MetadataValidatorDissertation` oder `MetadataValidatorBericht`, welche zur Validierung des Metadatensatzes der jeweiligen Dokumentenart verwendet werden. Dazu enthalten diese Klassen Validierungsinformationen über die einzelnen Elemente bzw. Sub-Elemente des Metadatensatzes, welche aus der entsprechenden Metadatensatzbeschreibung gewonnen werden. Objekte vom Typ `MetadataValidator` haben daher Zugriff auf ein Objekt vom Typ `MetadataSetData`, welches die Konfigurationsstruktur der Metadatensatzbeschreibung repräsentiert (also je nach Dokumentenart „`MetadataSetDissertation.xml`“ oder „`MetadataSetBericht.xml`“). Die Schnittstellenimplementierung `GenericMetadataSetData` repräsentiert dabei eine Metadatensatzbeschreibung für die Dokumentenart „Bericht“, die Spezialisierung `DissertationMetadataSetData` eine Metadatensatzbeschreibung für die Dokumentenart „Dissertation“, welche eine Obermenge der erstgenannten ist.

Ein Objekt vom Typ `MetadataSet` enthält Aggregatbeziehungen zu Objekten, die jeweils ein *Dublin Core*-Element (ein Element des Metadatensatzes) repräsentieren. Beispielhaft ist das in der Abbildung für das *Dublin Core*-Element „DC.Contributor“ gezeigt und ist für alle anderen Elemente ähnlich realisiert. Die Schnittstelle `DC_Contributor` beschreibt das Element allgemein und wird von der Klasse `DC_ContributorGeneric` implementiert, welche bereits alle Funktionalitäten realisiert, die für „DC.Contributor“-Elemente der Dokumentenart „Bericht“ benötigt werden. Die Spezialisierung `DC_ContributorDissertation` erweitert diese Klasse für die Dokumentenart „Dissertation“[6]. Jede dieser Klassen enthält intern weiter Aggregate, aus denen die Sub-Element-Struktur gebildet wird. Die dafür verwendeten Klassen befinden sich in der Regel in dem Package „`utilities`“, welches im Abschnitt 3.3.2.2 bereits vorgestellt wurde. Jedes Objekt vom Typ `MetadataSet` und jedes Objekt, das ein *Dublin Core*-Element beschreibt (wie beispielsweise Objekte des Typs `DC_Contributor`) implementieren darüberhinaus die Schnittstelle `XMLNode`, deren einzige Methode `createNode()` von den Klassen entsprechend überschrieben wird, so dass sie eine XML-Repräsentation liefert[7]. Die Klasse `SubElement` enthält alle Informationen, die zu einem Sub-Element aus der Metadatensatzbeschreibung über `MetadataSetData` gewonnen werden können, und stellt entsprechende Methoden wie etwa `getParameterName()`, `getLabel()`, `getDescription()`, `getMinFrequency()`, `getMaxFrequency()` usw. zur Verfügung. `ValidationInformation` enthält eine Datenstruktur, in der abgelegt ist, welche Elemente bzw. Sub-Elemente bereits wie validiert wurden.

[6] in diesem Fall um die Sub-Elemente „DC.Contributor.Advisor“ (Betreuer) und „DC.Contributor.Advisor“ (Gutachter), siehe dazu auch Abschnitt A.2 im Anhang

[7] d.h. die Klasse `MetadataSet` ruft in ihrer Implementation von `createNode()` die `createNode()`-Methode der Klasse jeder enthaltenen *Dublin Core*-Kategorie auf und setzt die Einzelstrukturen zu einer Gesamtstruktur zusammen

Die Aktionen des *MetadatenGenerator* auf der Ebene dieses Packages bestehen also darin, Parameter entgegenzunehmen, danach diejenigen herauszufiltern, die Informationen zu Metadaten enthalten[8] und diese Parameter mit Hilfe der zur jeweiligen `MetadataSet`-Klasse gehörenden `MetadataValidator`-Klasse zu validieren. Gleichzeitig wird das `MetadataSet`-Objekt (bzw. dessen Aggregate) aktualisiert, zusätzlich auch die Datenstruktur innerhalb von `ValidationInformation`. Dies geschieht solange, bis der Metadatensatz vollständig ist (im Sinne der primären Metadaten). Anschließend wird die zugehörige XML-Struktur erzeugt (durch die jeweiligen `createNode()`-Methoden) und temporär (dateibasiert oder datenbankbasiert) gespeichert.

[8] dies ist deshalb möglich, weil die Parameternamen in der Metadatensatzbeschreibung definiert sind

4.2. Realisierung

Dieser Abschnitt soll die Realisierung der ersten Teilanwendung *MetadatenGenerator* anhand von einigen „Screenshots“ beschreiben. Dabei wird der im obigen Sequenzdiagramm gezeigte Ablauf noch einmal verdeutlicht, gleichzeitig kann das Layout der Benutzeroberfläche demonstriert werden. Anhand der nächsten Abbildung wird bereits deutlich, dass das Layout der Teilanwendung aus dem Großen Beleg nahezu vollständig übernommen wurde[9]:

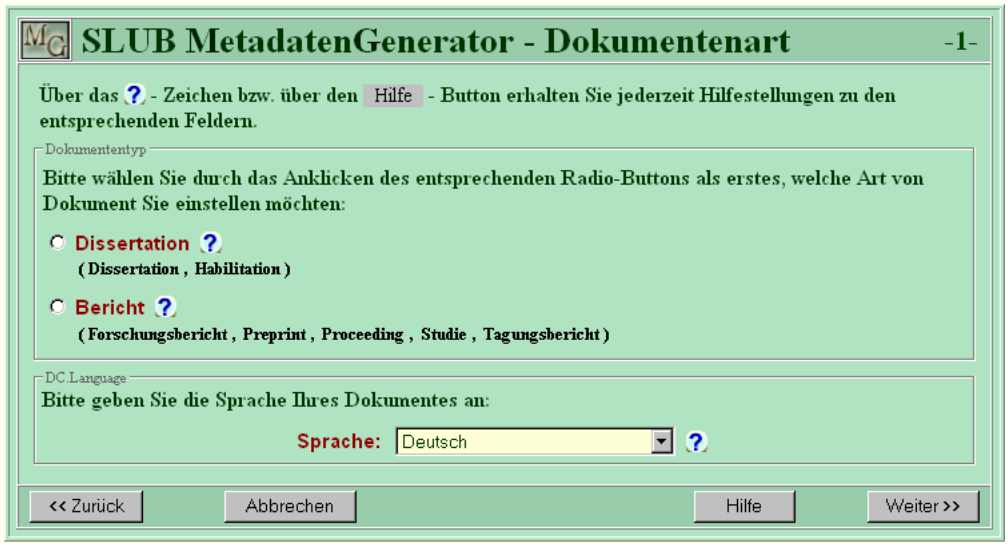
SLUB MetadatenGenerator - Dokumentenart
-1-
Über das ? - Zeichen bzw. über den Hilfe - Button erhalten Sie jederzeit Hilfestellungen zu den entsprechenden Feldern.
Dokumententyp
Bitte wählen Sie durch das Anklicken des entsprechenden Radio-Buttons als erstes, welche Art von Dokument Sie einstellen möchten:
Dissertation
(Dissertation , Habilitation)
Bericht
(Forschungsbericht , Preprint , Proceeding , Studie , Tagungsbericht)
DC.Language
Bitte geben Sie die Sprache Ihres Dokumentes an:
Sprache:
Deutsch
<< Zurück
Abbrechen
Hilfe
Weiter >>

Abbildung 34 *- SLUB MetadatenGenerator: Formular 1*

Nachdem der Autor die URL der *MetadatenGenerator*-Webanwendung angewählt hat, wird ihm zunächst ein Begrüßungsformular präsentiert, das hier nicht abgebildet ist. Über den „Weiter“-Button gelangt er dann zu dem in **Abbildung 34** gezeigten Formular. In allen Formularen kann er jederzeit über „Abbrechen“ die Teilanwendung verlassen. Klickt der Nutzer auf das *MetadatenGenerator*-Symbol in der linken oberen Ecke, gelangt er zu einem Konfigurationsformular, in dem er z. Bsp. einstellen kann, ob eine (zusätzliche) JavaScript-Validierung erfolgen soll (Standard) oder ob er eine E-Mail-Benachrichtigung wünscht (Standard). Außerdem kann er Teile des Layouts, wie etwa die Farbgestaltung etc. beeinflussen. Über die Buttons „Weiter“ und „Zurück“ ist jederzeit die Navigation innerhalb des *MetadatenGenerators* möglich. Dabei ist zu beachten, dass erfolgte Eingaben erst mit einem anschließendem Klicken auf „Weiter“ von der Anwendung übernommen werden. In jedem einzelnen Formular hat der Nutzer

[9] Hinweise zu den Gründen für dieses Design finden sich in[Schb00]

zudem die Möglichkeit, spezielle Hilfe zu bestimmten Themen über das Fragezeichen oder allgemeine Hilfestellungen über den „Hilfe“-Button zu erhalten. Ein Hilfefenster ist in der folgenden **Abbildung 35** zu sehen.

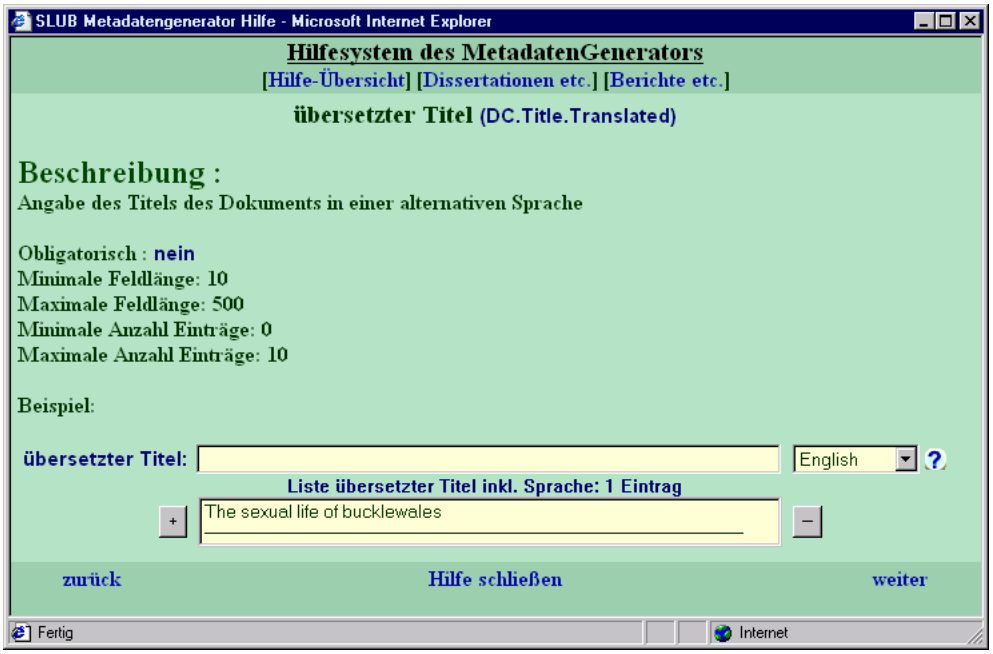

Abbildung 35 *- SLUB MetadatenGenerator: Hilfe*

Diese Abbildung zeigt die kontextsensitive Hilfe, wie sie sich bei einem Klick auf das Fragezeichen-Symbol hinter dem *Dublin Core*-Subelement „DC.Title.Alternative“ präsentieren würde. Es erfolgt jeweils eine kurze Beschreibung (die sich automatisch aus der Metadatensatzbeschreibung ergibt) sowie einige Angaben zu Attributen des jeweiligen Subelements (auch diese ergeben sich aus der Metadatensatzbeschreibung und passen sich daher eventuellen Änderungen automatisch an). Danach wird ein vollständiges Beispiel für das jeweilige Subelement in demselben Design gezeigt, wie es sich dem Autor auch innerhalb des *MetadatenGenerators* präsentiert. Desweiteren kann der Autor in der Hilfe unabhängig navigieren, um sich beispielsweise über weitere Subelements zu informieren, unter Umständen sogar über solche anderer Dokumentenarten.

Die Wahl einer der in **Abbildung 34** gezeigten Dokumentenarten durch den Autor muss zeitlich als erstes geschehen, da von dieser Wahl viele weitere (interne) Schritte des *MetadatenGenerators* abhängen[10]. Durch das Klicken auf „Weiter“ gelangt der Autor zum nächsten Formular, das in **Abbildung 36** gezeigt wird:

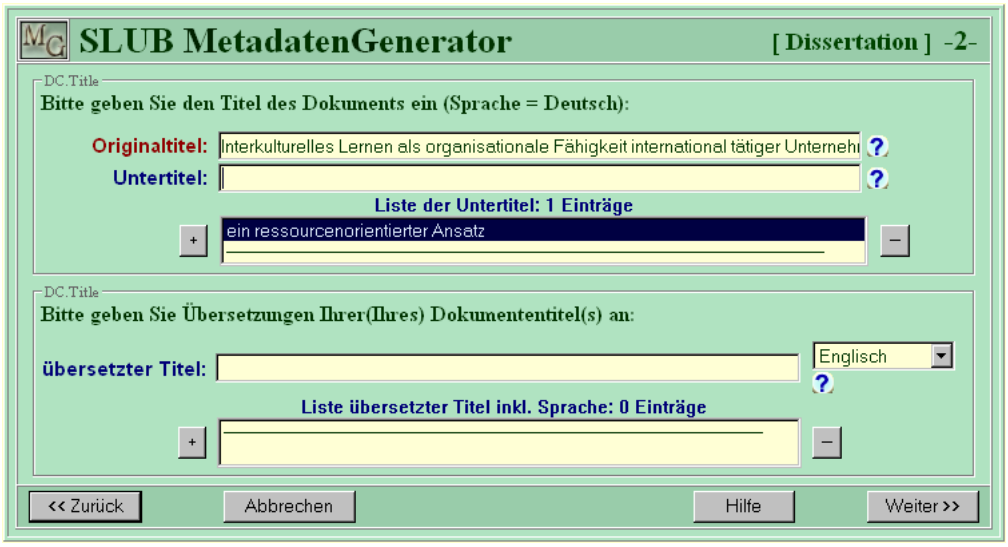

Abbildung 36 *- SLUB MetadatenGenerator: Formular 2*

Dies ist das erste einer Reihe von ähnlichen Formularen, mit deren Hilfe der Autor nach und nach seine Metadaten an das Gesamtsystem übermitteln kann. Die einzelnen Elemente bzw. Subelemente sind dabei sehr übersichtlich und zusammenhängend angeordnet. Dabei werden obligatorische bzw. optionale Felder durch unterschiedliche Farbgebung markiert. Die Eingaben des Autors werden durch JavaScript-Funktionen bereits validiert, sobald der „Weiter“-Button aktiviert wurde (also noch bevor die Daten an den *Controller* übermittelt werden), falls der Autor diese Voreinstellung nicht deaktiviert hat. Dieses Vorgehen hat den Vorteil, dass der Autor im Fehlerfall sofort informiert werden kann und somit die Chance hat, die Fehler zu berichtigen, bevor diese auf der Server-Seite erkannt werden, wo selbstverständlich auch noch einmal validiert wird. Dies erspart im Fehlerfall unnötige Kommunikation zwischen Client und Server. Hat der Autor alle relevanten Metadaten in die verschiedenen Formulare eingetragen, wird ihm noch einmal eine Übersicht aller Eingaben präsentiert, wie sie in **Abbildung 37** auf der folgenden Seite dargestellt ist.

[10] Einlesen der entsprechenden Metadatensatzbeschreibung, erstellen der korrekt spezialisierten Instanzen durch entsprechende „Fabriken“ usw.

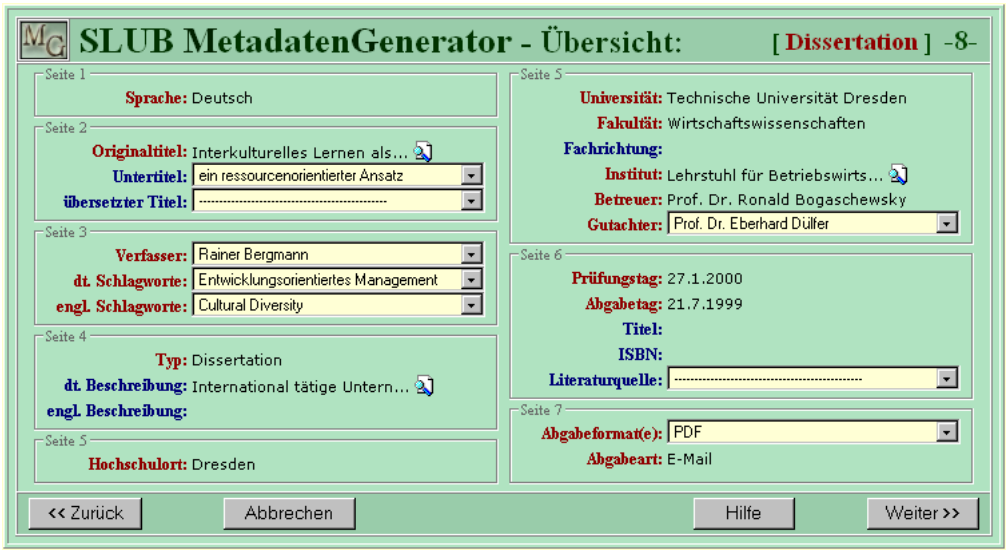

Abbildung 37 *- SLUB MetadatenGenerator: Übersicht*

Die Präsentation der Metadaten erfolgt geordnet nach den einzelnen Formularen (Seiten). Der Autor kann jederzeit Veränderungen vornehmen, da jedes einzelne Subelement verlinkt ist und ihn beim Anklicken direkt auf das entsprechende Formular bringt. Auch Texte, die aufgrund des begrenzten Platzes dieses Formulars nicht komplett dargestellt werden können, sind über entsprechende Zusatzaktionen (z. Bsp. den Klick auf ein Symbol) vollständig verfügbar. In einem weiteren, hier nicht abgebildeten Formular muss sich der Autor mit den rechtlichen Bedingungen des HSSS einverstanden erklären und kann einige persönliche Informationen (E-Mail, Telefonnummer etc.) für die SLUB-Mitarbeiter hinterlassen. Ein nochmaliger Klick auf „Weiter“ führt auf der Server-Seite dazu, dass noch einmal alle Metadaten des Autors auf Vollständigkeit[11] überprüft werden.

Konnte diese Überprüfung erfolgreich abgeschlossen werden, wird der Metadatensatz von der Anwendung in seine XML-basierte Repräsentation umgewandelt und anschließend temporär gespeichert. Die *Session* des Autors wird vom System beendet und es wird ihm eine Webseite präsentiert, welche die Übernahme der temporären Metadaten in das Gesamtsystem bestätigt. Gleichzeitig verschickt die Anwendung eine E-Mail an eine in den Konfigurationsstrukturen einstellbare Addresse[12], um die

[11] Vollständigkeit bedeutet in diesem Zusammenhang:
- alle primären obligatorischen Metadaten sind vorhanden und konnten korrekt validiert werden
- alle primären optionalen Metadaten, die übermittelt wurden, konnten auch korrekt validiert werden

[12] zur Zeit `hsss@slub-dresden.de`

SLUB-Mitarbeiter des Referats 5.3 vom Eingang eines neuen Metadatensatzes zu informieren. Hat der Autor die Voreinstellung, dass er eine E-Mail-Benachrichtigung erhält, nicht deaktiviert und gleichzeitig im letzten Formular seine E-Mail-Addresse bekanntgegeben, erhält er ebenfalls eine Bestätigungsnachricht, in der er auf weitere Möglichkeiten im Umgang mit dem Gesamtsystem hingewiesen wird (siehe dazu auch Abschnitt 4.4.2). Der Metadatensatz ist nun auch in der Teilanwendung *AdminTools* verfügbar, welche im Kapitel 5 ab Seite 145 beschrieben wird.

4.3. Persistenzrealisierung

Bei der Realisierung der temporären Persistenz von Metadatensätzen im Rahmen der Diplomarbeit wurde die dateibasierte Version des Belegs beibehalten, dazu aber um eine datenbankbasierte Implementierung ergänzt. Welche der beiden Möglichkeiten verwendet werden soll, kann über die Konfigurationsdatei „`web.xml`“ eingestellt werden[13]. Bei der dateibasierten Version wird die XML-Struktur des entsprechenden Metadatensatzes an die schon vorhandenen Metadatensatzstrukturen in einer wieder über die Konfigurationsstrukturen einstellbaren XML-Daten-Datei[14] angehängt.

Bei der Datenbank-Version wird die XML-Struktur des Metadatensatzes in einer durch die Konfigurationsstrukturen bestimmbare Tabelle gespeichert. Die Struktur des Datenmodells, das bis zu diesem Zeitpunkt betrachtet werden muss, ist dabei relativ flach (siehe dazu auch Anhang C ab Seite 259). Es existieren nur einige wenige Tabellen, die entweder Konfigurationsstrukturen beinhalten (wie etwa `config_hsss` oder `config_metadatasets`) oder aber die XML-Repräsentationen der Metadatensätze (`temp_metadatasets`). Letztere Tabelle (siehe dazu auch Abschnitt C.2 ab Seite 265) ist nachfolgende abgebildet:

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	Identifikationsnummer
processingid	char(100)	-	-	√	Bearbeitungsnummer (**unique**)
kindofdocument	varchar(30)	-	-	√	Dokumentenart
creationdate	datetime	-	-	√	Eingangsdatum
lastprocessingdate	datetime	-	-	√	Datum der letzten Bearbeitung
state	varchar(30)	-	-	√	aktueller Bearbeitungszustand
lastprocessor	varchar(100)	-	-	√	letzter Bearbeiter
metadataset	text	-	-	-	der XML-basierte Metadatensatz
metadatasetcopy	text	-	-	-	das XML-basierte Metadatensatz-Original des Autors

Tabelle 5 *- die Tabelle* `temp_metadatasets`

Wie der Tabelle zu entnehmen ist, wird die XML-Repräsentation des Metadatensatzes zu diesem Zeitpunkt direkt in die Spalte einer Tabelle geschrieben, es findet

[13] wobei die datenbankbasierte Version natürlich zu bevorzugen ist
[14] Eine solche XML-Datei existiert pro Dokumentenart

keine Partitionierung bzw. Normalisierung auf mehrere Tabellen entsprechend der inneren Struktur des Metadatensatzes statt. Das würde zu diesem Zeitpunkt auch keinen Sinn machen, da der Metadatensatz von der zweiten Teilanwendung *AdminTools* gegebenenfalls noch etliche Male iterativ geändert wird, was im Falle einer Normalisierung jedesmal den Aufwand bedeuten würde, die Gesamtstruktur durch multiple Joins aus der Datenbank holen zu müssen und anschließend wieder partitioniert zu speichern. Dies ist erst dann vernünftig, wenn die Bearbeitung des Metadatensatzes durch die *AdminTools*-Teilanwendung beendet ist und der Metadatensatz endgültig persistent eingestellt werden kann. Stattdessen wird die XML-Struktur also direkt in zwei Spalten der Tabelle abgelegt (`metadataset` und `metadatasetcopy`), so dass sie von der Anwendung später direkt mit Hilfe eines XML-Parsers ausgelesen und anschließend in eine äquivalente Objektstruktur umgeformt werden kann. Die Spalte `metadataset` beinhaltet dabei die aktuelle „Arbeitsversion“ des jeweiligen Metadatensatzes, die Spalte `mtadatasetcopy` hingegen den originalen Metadatensatz, so wie er vom Autor erstellt wurde, so dass im Falle eines Datenverlusts durch eine falsche Bearbeitung des Metadatensatzes dieser stets wieder auf den Ausgangszustand zurückgesetzt werden kann. Dadurch kann verhindert werden, dass die Primär-Metadaten des Autors noch einmal nachgefordert werden müssen. Desweiteren enthält die Tabelle eine Reihe weiterer Spalten, die jeweils Teile des Metadatensatzes (insbesondere der temporären Metadaten) direkt verfügbar machen, so dass eine Anwendung, die nur diese Informationen benötigt, diese nicht mühsam aus der XML-Struktur herausfinden muss und daher auch keine zusätzlichen Werkzeuge wie etwa einen XML-Parser benötigt.

4.4. Zusätzliche Funktionalitäten

Neben der im Abschnitt 4.2 vorgestellten Teilanwendung *MetadatenGenerator*, die alle wesentlichen Anforderungen erfüllt, gab es eine Reihe von weiteren Funktionalitäten, die in diesem Zusammenhang zumindest wünschenswert waren. Ein Teil dieser Zusatzfunktionalitäten konnte mit Hilfe zweier in der Abteilung Datenverarbeitung der SLUB beschäftigter Auszubildender schon im Rahmen der Diplomarbeit realisiert werden. Die folgenden Abschnitte sollen diese Realisierungen jeweils kurz beschreiben.

4.4.1. Evaluation

Am Ende des in Abschnitt 4.2 beschriebenen Vorgangs der Erstellung der Primär-Metadaten durch den Autor wird diesem bei erfolgreicher Speicherung eine Web-Seite mit einer Bestätigung präsentiert. Die Anwendung erlaubt es zusätzlich, in diese Seite ein Formular einzubinden, welches den Autor entscheiden lässt, ob er vor Beendigung der Teilanwendung noch eine bestimmte Ressource im Netz aufsuchen will. Ob eine solche Erweiterung angezeigt werden soll, ist in den Konfigurationsstrukturen einstellbar, daneben auch die URL der Ressource und die Beschriftung des Formulars. In der gegenwärtigen Version des *MetadatenGenerators* wird diese Möglichkeit dazu genutzt, ein Formular mit der Beschriftung „Fragebogen anzeigen“ und vorselektierter Checkbox in die Seite zu integrieren. Belässt der Autor die Selektion, führt ihn der Klick auf „Beenden“ zunächst auf eine Seite, auf der er die Teilanwendung *MetadatenGenerator* evaluieren kann (siehe **Abbildung 38** auf der folgenden Seite).

Diese Befragung erfolgt anonym und dient dazu, von den Autoren ein Feedback über die Qualität und Benutzerfreundlichkeit des *MetadatenGenerators* zu bekommen. Deaktiviert der Autor die Vorselektierung, kann er die Teilanwendung mit „Beenden“ normal verlassen. Auch wenn die Evaluation später einmal nicht mehr benötigt wird, kann diese Erweiterung durch die mögliche Konfigurierbarkeit sinnvoll genutzt werden. So kann die SLUB die URL und den Text ja beliebig ändern, um beispielsweise eine Webseite mit aktuellen Neuigkeiten rund um den HSSS (z. Bsp. „`http://hsss.slub.dresden.de/news.html`“) mit dem Text „letzte Neuigkeiten anzeigen“ zu lancieren.

SLUB Hochschulschriftenserver - MetadatenGenerator - Feedback
Liebe Autorin, lieber Autor,
für die optimale Gestaltung unserer MetadatenGenerator-Seiten möchten wir Sie um Ihre Meinung bitten.
Sie können die einzelnen Kategorien bewerten und/oder im Kommentarfeld Ihre Anregungen, Wünsche und Kritiken äußern.
Wie sind Sie auf die Möglichkeit der elektronischen Veröffentlichung aufmerksam geworden?
Prüfungsamt
Suchmaschine
Mund-zu-Mund-Propaganda
Zufall
anderer:
Gestaltung
Formale Gestaltung
-der Site sehr gut gut zufrieden schlecht
-der Formulare sehr gut gut zufrieden schlecht
Inhaltliche Gestaltung
-der Site sehr gut gut zufrieden schlecht
-der Formulare sehr gut gut zufrieden schlecht

Abbildung 38 *- SLUB MetadatenGenerator: Fragebogen (Ausschnitt)*

4.4.2. Status-Verfolgung durch den Autor

Eine weitere wünschenswerte Funktion aus Sicht des Autors ist sicherlich die, über den Stand der Bearbeitung seines Metadatensatzes aktuell informiert zu sein. Zu diesem Zweck existiert eine Anwendung, die es ihm erlaubt, sich die aktuellen Statusinformationen für seinen Metadatensatz anzeigen zu lassen. Dazu muss der Anwendung selbstverständlich die Bearbeitungsnummer mitgeteilt werden, die den Metadatensatz eindeutig identifiziert.

In der E-Mail, die der Autor gegebenenfalls als Bestätigung erhält, ist unter anderem gleich ein Link enthalten, der die Bearbeitungsnummer integriert und nur noch angeklickt werden muss, um sofort die aktuellen Statusinformationen des entsprechenden Metadatensatzes anzuzeigen. Eine solche Statusinformation präsentiert sich dem Autor wie in **Abbildung 39** auf der nächsten Seite dargestellt. Um diese Informationen anzuzeigen, verwendet die Anwendung die Spalten der bereits vorgestellten Tabelle `temp_metadatsets`, um direkten Zugang zu den Statusinformationen wie „Bearbeitungsstatus“, „Datum der letzten Bearbeitung“ oder „letzter Bearbeiter“ zu erhalten und nicht einen XML-Parser bemühen zu müssen.

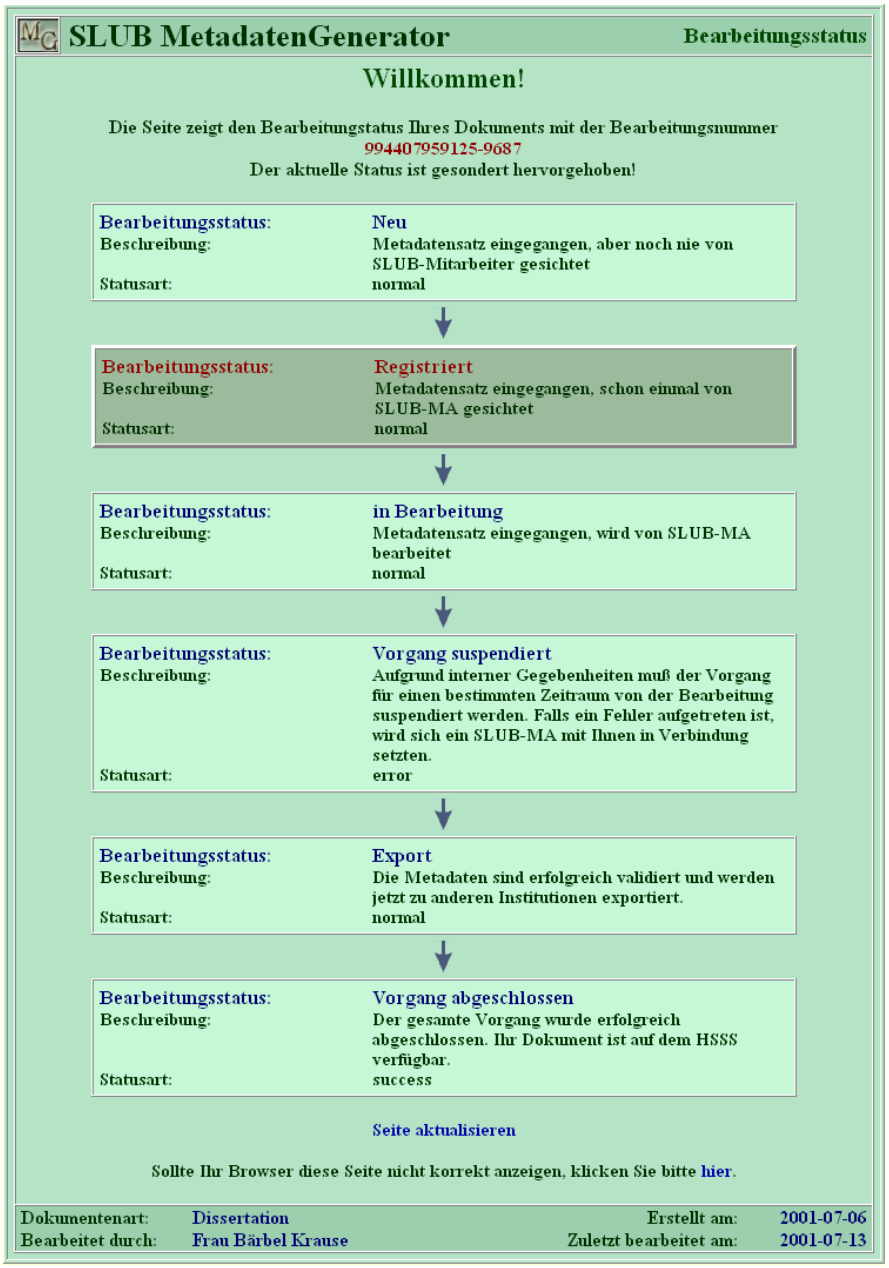
SLUB MetadatenGenerator
Bearbeitungsstatus
Willkommen!
Die Seite zeigt den Bearbeitungstatus Ihres Dokuments mit der Bearbeitungsnummer
99440795912 5-9687
Der aktuelle Status ist gesondert hervorgehoben!
Bearbeitungsstatus: Neu
Beschreibung: Metadatensatz eingegangen, aber noch nie von SLUB-Mitarbeiter gesichtet
Statusart: normal
Bearbeitungsstatus: Registriert
Beschreibung: Metadatensatz eingegangen, schon einmal von SLUB-MA gesichtet
Statusart: normal
Bearbeitungsstatus: in Bearbeitung
Beschreibung: Metadatensatz eingegangen, wird von SLUB-MA bearbeitet
Statusart: normal
Bearbeitungsstatus: Vorgang suspendiert
Beschreibung: Aufgrund interner Gegebenheiten muß der Vorgang für einen bestimmten Zeitraum von der Bearbeitung suspendiert werden. Falls ein Fehler aufgetreten ist, wird sich ein SLUB-MA mit Ihnen in Verbindung setzten.
Statusart: error
Bearbeitungsstatus: Export
Beschreibung: Die Metadaten sind erfolgreich validiert und werden jetzt zu anderen Institutionen exportiert.
Statusart: normal
Bearbeitungsstatus: Vorgang abgeschlossen
Beschreibung: Der gesamte Vorgang wurde erfolgreich abgeschlossen. Ihr Dokument ist auf dem HSSS verfügbar.
Statusart: success
Seite aktualisieren
Sollte Ihr Browser diese Seite nicht korrekt anzeigen, klicken Sie bitte hier.
Dokumentenart: Dissertation
Bearbeitet durch: Frau Bärbel Krause
Erstellt am: 2001-07-06
Zuletzt bearbeitet am: 2001-07-13

Abbildung 39 *- SLUB MetadatenGenerator: Bearbeitungsstatus*

Wie die Abbildung zeigt, wird dem Autor die gesamte Abfolge der Bearbeitungszustände in ihrer zeitlichen Reihenfolge gezeigt. Der für seinen Metadatensatz momentan zutreffende Bearbeitungsstatus wird dabei gesondert hervorgehoben. So kann der Autor die augenblickliche Situation relativ gut einschätzen. Zu den weiteren interessanten Informationen gehören der Name des Bearbeiters sowie das Datum der letzten Bearbeitung. Der Vorteil in dieser zusätzlichen Fuktionalität liegt für die Mitarbeiter des

Referats 5.3 vor allem darin, dass der Autor eine bequeme Möglichkeit hat, sich über den Bearbeitungsstatus online zu informieren und damit für ihn nicht die Notwendigkeit einer direkten Interaktion (telefonisch, per E-Mail) besteht, was wiederum die Mitarbeiter des Referats 5.3 entlastet.

4.4.3. MetadatenGenerator Light - eine Alternativ-Version

Auch wenn mit der bisher vorgestellten Version des *MetadatenGenerators* alle Anforderungen an eine solche Anwendung erfüllt werden konnten, bleiben einige wenige Benutzer ausgeschlossen, nämlich solche, die die technischen Voraussetzungen für die Nutzung dieser Anwendung auf der Client-Seite nicht erfüllen können. Diese wesentlichen clientseitigen Voraussetzungen (an den Browser) sind:

- Unterstützung von JavaScript (mindestens Version 1.2)
- Unterstützung weiter Teile von CSS (Cascading Stylesheets) Version 1.0
- Unterstützung weiter Teile von HTML 4.0

Diese Anforderungen werden im allgemeinen von allen weitverbreiteten Browsern ab der Version 4.0 unterstützt, womit zur Zeit etwa 95% aller verwendeten Browser abgedeckt ist, so dass in etwa auch dieselbe Prozentzahl potentieller Nutzer durch die Anwendung unterstützt werden kann. Die restlichen fünf Prozent der Nutzer unterteilen sich in Nutzer,

- die zwar einen modernen Browser benutzen, jedoch aus Sicherheitsgründen Technologien wie JavaScript nicht verwenden (deaktiviert haben)
- die immer noch sehr alte Browser verwenden, im Extremfall zeichenorientierte Browser wie etwa Lynx

Um auch diese Gruppe nicht von einer Nutzung auszuschließen, bedarf es einer alternativen Teilanwendung, die zwar möglichst eine annähernde Funktionalität wie der *MetadatenGenerator* bietet, dabei allerdings nur minimale Ansprüche an die Fähigkeiten der Clients (Browser) stellt. Eine solche Version wurde in Zusammenarbeit mit den bereits erwähnten Auszubildenden der Abteilung Datenverarbeitung der SLUB

unter dem Namen *„MetadatenGenerator Light“* erfolgreich entwickelt. Die alternative Version verlangt von einem Browser lediglich die Unterstützung von HTML 2.0 und konnte sogar auf textorientierten Browsern wie Lynx getestet werden. Dabei wurden Modifikationen natürlich in erster Linie bei den *View*-Komponenten in Form von separaten JSP's notwendig, *Controller* und *Model* brauchten aufgrund des sorgfältigen Entwurfes nur geringfügig angepasst zu werden.

Vom Ablauf der Aktionen ähneln sich die beiden Versionen ebenfalls stark, denn es wurde darauf Wert gelegt, das Layout und die Benutzerfreundlichkeit der Originalversion so weit wie möglich zu erhalten, auch wenn aufgrund der geringen Möglichkeiten des Clients natürlich an vielen Stellen Kompromisse gemacht werden mussten. So präsentiert die alternative Version ebenfalls als erstes eine Begrüßungsseite, bevor ein Formular mit den verfügbaren Dokumentenarten gezeigt wird. Hat der Autor sich für eine bestimmte Dokumentenart entschieden, wird allerdings ein einziges großes Formular für alle vom Autor zu übermittelnden Metadaten generiert, welches ausschnittsweise in **Abbildung 40** zu sehen ist:

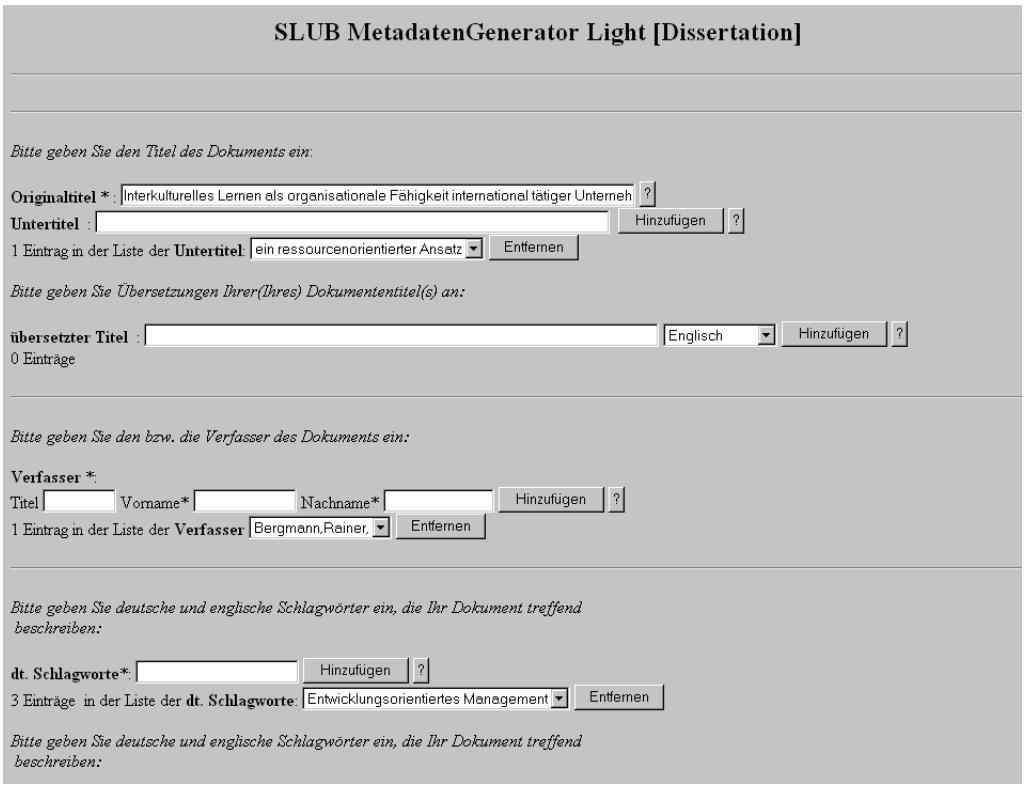
SLUB MetadatenGenerator Light [Dissertation]
Bitte geben Sie den Titel des Dokuments ein:
Originaltitel * : Interkulturelles Lernen als organisationale Fähigkeit international tätiger Unterneh ?
Untertitel : Hinzufügen ?
1 Eintrag in der Liste der Untertitel: ein ressourcenorientierter Ansatz Entfernen
Bitte geben Sie Übersetzungen Ihrer(Ihres) Dokumententitel(s) an:
übersetzter Titel : Englisch Hinzufügen ?
0 Einträge
Bitte geben Sie den bzw. die Verfasser des Dokuments ein:
Verfasser *:
Titel Vorname* Nachname* Hinzufügen ?
1 Eintrag in der Liste der Verfasser Bergmann,Rainer, Entfernen
Bitte geben Sie deutsche und englische Schlagwörter ein, die Ihr Dokument treffend beschreiben:
dt. Schlagworte*: Hinzufügen ?
3 Einträge in der Liste der dt. Schlagworte: Entwicklungsorientiertes Management Entfernen
Bitte geben Sie deutsche und englische Schlagwörter ein, die Ihr Dokument treffend beschreiben:

Abbildung 40 *- SLUB MetadatenGenerator Light (Ausschnitt)*

Dieses Formular ist vom Autor nach und nach abzuarbeiten, wobei trotz der geringen Anforderungen an den Browser eine gewisse Benutzerfreundlichkeit realisiert werden konnte, indem immer wieder an die korrekten Stellen innerhalb des Formulars gesprungen wird. Außerdem ist auch für diese Version eine kontextsensitive Hilfe verfügbar, welche ebenfalls separat navigierbar ist. Nachdem der Autor das Formular vollständig ausgefüllt hat, kann er es durch ein Klicken auf „Weiter“ komplett validieren lassen. Hat er den rechtlichen Bedingungen zugestimmt, wird der Metadatensatz temporär gespeichert und der Autor erhält auch hier eine Bestätigungsnachricht. Auch für die alternative Version existiert ein Fragebogen, der vom Autor ausgefüllt werden kann, falls er das wünscht.

4.5. Einschätzung

Die Teilanwendung *MetadatenGenerator* ist aufgrund der Tatsache, dass sie schon in der Belegarbeit umfassend implementiert wurde und im Rahmen der Diplomarbeit noch einmal nahezu vollständig neu realisiert wurde, von allen Teilanwendungen des Gesamtsystems die am besten getestete. Sie wird auch am ehesten allen Anforderungen an eine Produktiv-Version[15] gerecht und bietet eine Reihe sehr nützlicher zusätzlicher Funktionalitäten. Dabei geht die Teilanwendung in vielen Bereichen über die Anforderungen an eine solche Schnittstelle weit hinaus. Dazu zählt insbesondere die Möglichkeit, ohne Programmierkenntnisse weite Teile des Verhaltens und des Designs des *MetadatenGenerators* bestimmen zu können, vor allem die Präsentation der Metadatenelemente [16] sowie deren Behandlung bei der clientseitigen und Serverseitigen Validierung [17] (siehe auch Abschnitt B im Anhang). Darüber hinaus wurden im Abschnitt 3.2.1 noch weitergehende Konzepte vorgestellt, welche auf den vorhandenen Fähigkeiten der Teilanwendung aufbauen und diese noch wesentlich erweitern können. Trotzdem finden sich natürlich auch für den *MetadatenGenerator* eine Reihe von Verbesserungsmöglichkeiten bzw. sinnvolle Erweiterungen. Diese möglichen Verbesserungen werden in Kapitel 8.2 noch einmal strukturiert zusammengefasst und sollen hier nur kurz aufgezählt und begründet werden:

- Zur Zeit übernimmt der *MetadatenGenerator* Daten erst durch das Klicken auf den „Weiter“-Button[18]. Hat ein Nutzer ein Formular bereits ausgefüllt und klickt anschließend aber auf „Zurück“, gehen die eben eingetragenen Daten verloren. Dieser Zustand kann mit einigem Aufwand verbessert werden, so dass auch bei „Zurück“ die Daten übernommen werden, die dann aber noch nicht validiert werden sollten (weder clientseitig noch Server-seitig).

- Es wäre von Vorteil, wenn in jedem Formular eine minimierte Site-Map (vielleicht als dynamisches Symbol) zur Orientierung zur Verfügung stände. Außerdem könnte man dann in Sprüngen navigieren, was insbesondere bei der Korrektur von Formularen von Vorteil ist, um schnell wieder auf das Übersichtsformular zu gelangen.

[15] es muss aber an dieser Stelle noch einmal darauf hingewiesen werden, dass es **nicht** Ziel dieser Diplomarbeit war, produktive Versionen für die SLUB zu erstellen, sondern auf eine prototypische Art und Weise bestimmtes Know How zu sammeln und entsprechend zu demonstrieren.

[16] farbliche Gestaltung in Abhängigkeit ihrer Notwendigkeit, Beschriftung und Kurzhilfe usw.

[17] Notwendigkeit (obligatorisch, optional), minimales und maximales Auftreten, Mindest- und Höchstlänge etc.

[18] Die Gründe sind technischer Natur und stehen im Zusammenhang mit der Server-seitigen Validierung.

- Die im vorangegangenen Kapitel bereits angesprochenen TagLibs wären insbesondere auch für den Einsatz innerhalb der View des *MetadatenGenerators* geeignet, so dass Änderungen am Layout etc. noch einfacher und ohne weitere Kenntnisse zu realisieren wären.
- Die kontextsensitive Hilfe erhält bisher nur einen Teil ihrer Informationen aus den Konfigurationsstrukturen. Ein anderer Teil wird in Form von integrierten JSP's verwendet. Idealerweise würden diese Informationen in die Datenbank ausgelagert, was eine zusätzliche Anwendung zur transparenten Änderung dieser Inhalte durch Mitarbeiter des Referates 5.3 notwendig macht.

5. Teilanwendung 2 : AdminTools

Inhalt des Kapitels:

5.1. Analyse und Design 146
5.2. Realisierung 151
5.2.1. Unterstützung der Arbeitsabläufe 151
5.2.2. Administrationsunterstützung 162
5.3. Persistenzrealisierung 166
5.4. Einschätzung 174

5.1. Analyse und Design

Viele Aussagen zur objektorientierten Analyse und zum Design dieser Teilanwendung wurden von vorangegangenen Kapiteln bzw. Abschnitten bereits getroffen (siehe dazu insbesondere die Abschnitte 3.2.1, 3.3.2.2 und 4.1) und sollen deshalb an dieser Stelle nicht noch einmal wiederholt werden. Stattdessen soll der Abschnitt Besonderheiten des Entwurfs der Teilanwendung *AdminTools* näher beleuchten. Diese Besonderheiten ergeben sich zumeist aus der primären Aufgabe dieser Teilanwendung, die darin besteht, die in Abschnitt 3.1 spezifizierten Arbeitsabläufe technisch so gut wie möglich zu begleiten. Dazu muss die Anwendung bestimmte interne und externe Zustände kennen und entsprechend behandeln können. Diese Zustände werden in der Konfigurationsstruktur „`hsssdata.xml`“ definiert und stehen der Anwendung damit zur Verfügung. Zu den internen[1] Zuständen gehören solche, die sich auf die Metadaten der Hochschulschrift und solche, die sich auf die Dokumente[2] selbst beziehen. **Tabelle 6** zeigt die für Dokumente typischen Zustände:

Zustand	Zustandsbeschreibung
Dokument nicht vorhanden	Ausgangszustand. Das Dokument ist noch nicht vorhanden.
Dokument vorhanden	Das Dokument ist vorhanden, befindet sich physisch auf einem der Rechner der SLUB-Mitarb. des Referats 5.3.
Validierung Dokument	Die elektronische Version des Dokumentes wird mit dem Druckexemplar (stichprobenartig) verglichen.
Validierung Dokument OK	Die elektronische Version des Dokumentes konnte korrekt validiert werden.
Validierung Dokument Fehler	Die elektronische Version des Dokumentes konnte nicht korrekt validiert werden. Das erhaltene Dokument war fehlerhaft.
Konvertierung Dokument	Das Format des Dokumentes wird in das Zielformat (HSSS-Präsentationsformat) konvertiert.
Konvertierung Dokument OK	Die elektronische Version des Dokumentes konnte korrekt konvertiert werden.
Konvertierung Dokument Fehler	Die elektronische Version des Dokumentes konnte nicht korrekt konvertiert werden. Das erhaltene Dokument war fehlerhaft.
Dokument OK	Endzustand für Dokumente. Dokument ist validiert, konvertiert und liegt im vorgesehenen Verzeichnis (aber noch nicht im Frontdoor-Verzeichnis).

Tabelle 6 *- Dokument-spezifische Zustände*

[1] nicht nach außen (außerhalb der *AdminTools*) sichbar
[2] die elektronische Version der Hochschulschrift (in möglicherweise mehreren Formaten)

Die Zustände begleiten die wesentlichen sich aus den spezifizierten Arbeitsabläufen ergebenden Aktionen im Zusammenhang mit Dokumenten: das formale, stichprobenartige Vergleichen mit der entsprechenden Druckversion und das Konvertieren in das entsprechende Präsentationsformat. **Tabelle 7** zeigt die für die Metadaten der Hochschulschrift relevanten Zustände:

Zustand	Zustandsbeschreibung
Validierung Primärmetadaten	Ausgangszustand. Metadatensatz wird von SLUB-Mitarbeitern auf Vollständigkeit und Richtigkeit der primären Metadaten überprüft.
Validierung Primärmetadaten OK	Die primären Metadaten konnten korrekt validiert werden.
Validierung Primärmetadaten Fehler	Schwerer technischer Fehler. Die Primärmetadaten konnten nicht korrekt validiert werden.
Validierung Fachreferenten	Metadatensatz wird jetzt von Fachreferenten bearbeitet, welche zusätzliche (sekundäre) Metadaten wie Klassifikationen u. Schlagworte aus Schlagwortnormdateien hinzufügen.
Validierung Sekundärmetadaten	Die sekundären Metadaten werden von SLUB-Mitarbeitern validiert. Der gesamte Metadatensatz wird auf die Einstellung in die persistente Datenbasis vorbereitet.
Validierung Sekundärmetadaten OK	Endzustand für Metadaten. Die sekundären Metadaten konnten korrekt validiert werden.
Validierung Sekundärmetadaten Fehler	Fehler. Die sekundären Metadaten konnten nicht korrekt validiert werden.

Tabelle 7 *- Metadaten-spezifische Zustände*

Diese Zustände spiegeln den sich auf die Metadaten beziehenden Teil der Arbeitsabläufe aus Abschnitt 3.1 wieder: Primärmetadaten des Autors validieren, diese an den entsprechenden Fachreferenten weiterleiten, die von ihm erstellten Sekundär-Metadaten eingeben und validieren. Neben diesen beiden internen Zustandsarten existiert zusätzlich ein nach außen sichtbarer Gesamtzustand, der sich natürlich (zumindest teilweise) aus den beiden internen Zustandsarten ergibt. Dieser Gesamtzustand ist z. Bsp. durch die im Abschnitt 4.4.2 beschriebene Anwendung sichtbar, welche dem Autor den aktuellen Bearbeitungsstatus seiner Hochschulschrift mitteilt. Die Gesamtzustände dieser Teilanwendung zusammen mit vorauszusetzenden internen Zuständen (Dokumentenzustände bzw. Metadatenzustände) sind in **Tabelle 8** auf der folgenden Seite abgebildet.

Gesamtzustand	Metadatenzustand	Dokumentenzustand
Neu	-	-
Registriert	-	-
in Bearbeitung	Validierung Primärmetadaten OK	-
Vorgang suspendiert	Validierung Primärmetadaten Fehler (nicht durch Referat 5.3 reparierbar)	Validierung Dokument Fehler Konvertierung Dokument Fehler (Dokument lässt sich überhaupt nicht konvertieren)
Export	Validierung Sekundärmetadaten OK	-
Vorgang abgeschlossen	Validierung Sekundärmetadaten OK	Dokument OK

***Tabelle 8** - Gesamtzustände und Abhängigkeiten*

Neben den hier gezeigten Zuständen gibt es noch weitere Parameter, die auf das Ablaufverhalten der Teilanwendung einen Einfluss haben, durch die internen Zustände der zwei existierenden Kategorien aber nicht beschrieben werden können. Als Beispiel dafür lässt sich der letzte Gesamtzustand „Vorgang abgeschlossen“ verwenden. Voraussetzung für diesen Zustand sind wie gezeigt der Metadatenzustand „Validierung Sekundärmetadaten OK“ und der Dokumentenzustand „Dokument OK“, was sicherlich einsichtig ist, da ein korrektes Dokument im Präsentationsformat und ein vollständiger Metadatensatz Voraussetzung für ein Einstellen und Veröffentlichen der Hochschulschrift ist. Daneben gibt es allerdings noch weitere aus dem jeweiligen Geschäftsmodell der Dokumentenart ableitbare Bedingungen. So sind beispielsweise für die Dokumentenart „Dissertation“ mit dem Metadaten-Export zum SWB (*Südwestdeutscher Bibliotheksverbund*) und zur DDB (*Die Deutsche Bibliothek*) zwei weitere Voraussetzungen gegeben. Diese betreffen (ändern) aber weder den Zustand von Dokumenten, noch den von Metadaten. Aus diesem Grund werden diese Bedingungen nicht innerhalb der Zustandbeschreibung, sondern in einem separaten Bereich der Konfigurationsstruktur „`hsssdata.xml`“ beschrieben, der sich speziell auf den Metadatenexport bezieht (siehe auch Abschnitt B im Anhang). **Listing 8** zeigt auf der folgenden Seite den entsprechenden Ausschnitt aus „`hsssdata.xml`“. Hier sind alle Export-Institutionen (SWB, DDB) und die für den HSSS relevanten Parameter aufgeführt. Diese beschreiben, auf welche Art (`target`) der Export durchgeführt wird und welche Identifikationen die Teilanwendung bei den jeweiligen Institutionen (`entity-name`) dafür benötigt (`publisher-id`, `publisher-name`, `user` und `passwd`). Von der Anwendung wird für den Exportvorgang (für jede Exporteinrichtung) eine separate Zustandskategorie eingerichtet. Mögliche Zustände sind „nicht bekannt“, „Metadaten exportiert“, „Metadaten nicht exportiert“ und „OK-Rückmeldung“. Erfolgt ein Export an eine der Institutionen, wird deren Exportzustand automatisch auf „Metadaten exportiert“ gesetzt. Zusätzlich wird von der Anwendung der Export-Zeitpunkt

vermerkt. Der Parameter `delay` gibt an, wie viele Tage die SLUB-Mitarbeiter des Referats 5.3 möglichst auf eine Export-Bestätigung von der betreffenden Institution warten sollen, bevor sie den Metadatensatz endgültig einstellen[3].

```
<?xml version="1.0" encoding="ISO−8859−1" ?>
<HsssData:HSSSDATA xmlns:HsssData = "http://hsss.slub−dresden.de/xml/HsssData/">
...
   <HsssData:EXPORT>
      <HsssData:ExportEntity entity−name="ddb" publisher−id="L60000589" publisher−name="slub"
         target="ep@ddb.de" delay="14" user="" passwd="xxxxxxxx"/>
      <HsssData:ExportEntity entity−name="swb" publisher−id="DDSU" publisher−name="hsss"
         target="ftp.bsz−bw.de/daten/opus/DDSU/opus_neu" delay="14" user="DDSU" passwd="xxxxxxxx" />
   </HsssData:EXPORT>
...
</HsssData:HSSSDATA>
```

***Listing 8** : Export-Bereich der Konfigurationsstruktur „hsssdata.xml“*

Auch für den Export ist eine Bedingung zu erfüllen, nämlich dass die zu exportierenden Metadaten vollständig sind, was bei Erreichen des Metadatenzustandes „Validierung Sekundärmetadaten OK“ gegeben ist. Erst dann wird ein Export innerhalb der *AdminTools* zugelassen. Die vollständigen Voraussetzungen für ein mögliches Einstellen eines Metadatensatzes in die persistente Datenbasis des Gesamtsystems, angezeigt durch den Gesamtzustand „Vorgang abgeschlossen“, können jetzt also wie folgt angegeben werden:

- **für die Dokumentenart „Dissertation“:**
  - Metadatenzustand: „Validierung Sekundärmetadaten OK“
  - Dokumentenzustand: „Dokument OK“
  - Exportzustand DDB: „Metadaten exportiert“ bzw. „OK-Rückmeldung“
  - Exportzustand SWB: „Metadaten exportiert“ bzw. „OK-Rückmeldung“
- **für die Dokumentenart „Bericht“:**
  - Metadatenzustand: „Validierung Sekundärmetadaten OK“
  - Dokumentenzustand: „Dokument OK“
  - Exportzustand SWB: „Metadaten exportiert“ bzw. „OK-Rückmeldung“

Wünschenswert, aber eben nicht notwendig wäre es, dass mit der Einstellung des Metadatensatzes solange gewartet wird, bis der Exportzustand für jeden benötigten Export dem Zustand „OK-Rückmeldung“ entspricht.

[3]Bei dem Erhalt einer solchen Bestätigung würde der Bearbeiter den Exportzustand auf „OK-Rückmeldung“ setzen. Dafür würde die Anwendung auch wieder den Zeitpunkt mitprotokollieren.

Zu den weiteren Besonderheiten des Entwurfes der Teilanwendung *AdminTools* gehört auch die Integration einer Reihe von externen API's, die notwendig sind, um die gewünschten Funktionalitäten der Anwendung zu realisieren. So werden etwa der E-Mail-basierte Metadatenexport nach DDB oder der FTP-basierte Export nach SWB mit Hilfe solcher in die Anwendung integrierter Bibliotheken realisiert, die hier allerdings nicht im Detail beschrieben werden sollen[4].

[4]Einige Informationen zu den verwendeten API's finden sich im Anhang im Kapitel D

5.2. Realisierung

Dieser Abschnitt soll die Realisierung der Teilanwendung *AdminTools* anhand von einigen „Screenshots“ näher erläutern. Der Abschnitt unterteilt sich dabei in zwei Bereiche, im ersten wird der Teil der *AdminTools* beschrieben, der die Arbeitsabläufe der Geschäftsmodelle nachbildet. Der zweite Teil bezieht sich auf die Bereiche, die administrative Tätigkeiten in Bezug auf das Gesamtsystem unterstützen. Auch die Teilanwendung *AdminTools* verwendet zum größten Teil das in der Belegarbeit erstellte Layout, welches in [Schb00] näher beschrieben ist.

5.2.1. Unterstützung der Arbeitsabläufe

Die Teilanwendung *AdminTools* hat einen großen Einfluss auf das Gesamtsystem und insbesondere den Zugriff auf alle Metadaten des Systems, so dass sie nur einem ausgewählten Personenkreis[5] zur Verfügung stehen sollte. Außerdem sollten die Inhalte, die von dieser Anwendung über das Netz übertragen werden, grundsätzlich verschlüsselt werden, was durch einen SSL-fähigen Webserver geschehen kann. Um den Zugang zu dieser Teilanwendung zu schützen, wurde zum einen die Nutzerauthentifikation des Webservers genutzt, zum anderen muss sich ein Nutzer aber auch bei der Teilanwendung direkt anmelden. Im nicht angemeldeten Zustand ist nahezu keine Interaktion möglich. Hat sich der Nutzer hingegen erfolgreich angemeldet, werden ihm automatisch alle temporären Metadatensätze innerhalb des Gesamtsystems angezeigt, wie die **Abbildung 41** auf der folgenden Seite zeigt[6].

[5] zur Zeit ausschließlich den entsprechenden Mitarbeitern des Referats 5.3

[6] Ist ein Nutzer angemeldet, kann er diese Seite jederzeit auch durch das Klicken von „Metadatensätze“ in der oberen Navigationsleiste der *AdminTools* erreichen.

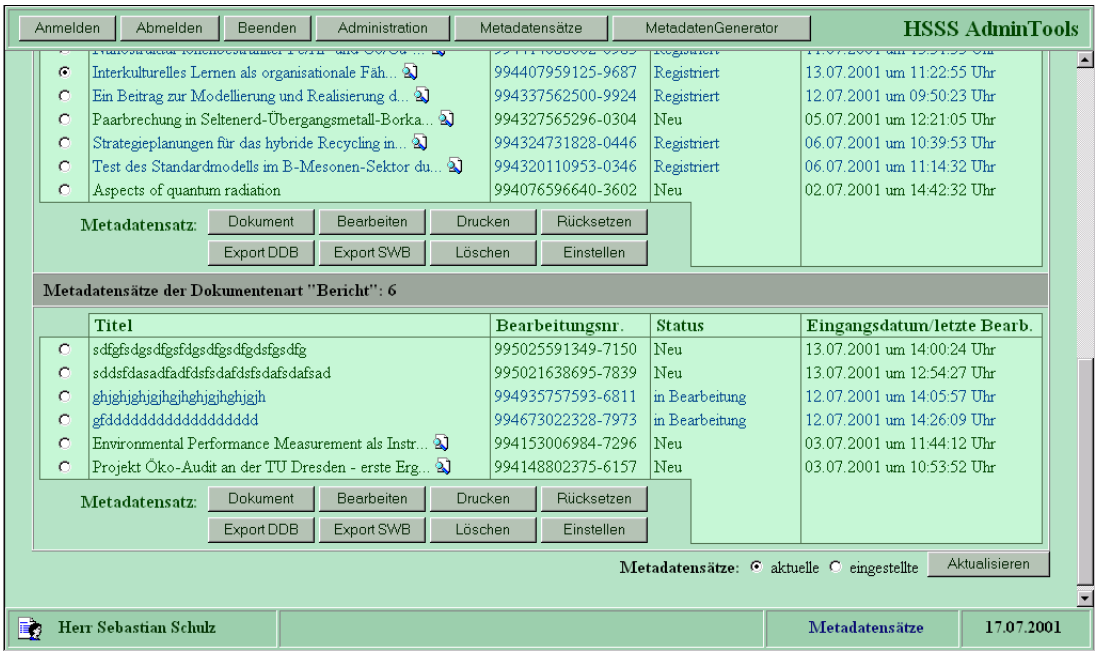

Abbildung 41 *- HSSS AdminTools: Übersicht der Metadatensätze*

Die Metadatensätze sind geordnet in der Reihenfolge ihres Eintreffens bzw. der letzten Bearbeitung. Sie verbleiben solange in dieser Ansicht, bis sie durch die *AdminTools* endgültig persistent eingestellt wurden[7]. Dabei werden, wie aus der Abbildung ersichtlich wird, die einzelnen Dokumentenarten unterschieden und die zugehörigen Metadatensätze getrennt aufgeführt. Die jeweilige Übersichts-Tabelle enthält natürlich nur einige ausgewählte Attribute, dazu gehören der Titel der Hochschulschrift, die Bearbeitungsnummer, die jeden Metadatensatz eindeutig identifiziert, der Gesamtstatus bzw. Gesamtzustand sowie das Eingangsdatum (im Falle eines neuen Metadatensatzes) oder das Datum der letzten Bearbeitung. Die verschiedenen Gesamtzustände der Metadatensätze werden dabei von der Anwendung auch farblich hervorgehoben. Jeder Metadatensatz kann selektiert werden, um ihn anschließend durch das Klicken auf einen der acht für die Bearbeitung vorhandenen Buttons entsprechend zu behandeln. In **Tabelle 9** auf der nächsten Seite werden diese Buttons und ihre Bedeutung näher erläutert. Auf der in **Abbildung 41** gezeigten Seite der *AdminTools* können aber auch bereits eingestellte Metadatensätze auf dieselbe Art und Weise behandelt werden, indem der Radiobutton im unteren Teil der Abbildung von „aktuelle“ auf „eingestellte“ Metadatensätze verändert und anschließend auf „Aktualisieren“ geklickt wird. Auch hier sind im Prinzip alle im folgenden beschrieben Aktionen anwendbar, allerdings mit einigen Ausnahmen, die im weiteren Verlauf erläutert werden.

[7] also vom Eintreffen, direkt nachdem die Autoren sie mit Hilfe des *MetadatenGenerators* erstellt haben, bis zum finalen Gesamtzustand „Vorgang abgeschlossen“

Button	Funktion
Dokument	Zeigt alle dokumentenspezifischen Informationen für den jeweiligen Metadatensatz an. So kann hier auch der Dokumenten-Zustand des Metadatensatzes verändert werden.
Bearbeiten	Hier können alle Metadaten des Metadatensatzes bearbeitet, hinzugefügt und validiert werden. Einer der zentralen Bereiche dieser Teilanwendung.
Drucken	Es erfolgt eine druckfreundliche Präsentation von Teilen der Metadaten, die somit gedruckt und an den entsprechenden Fachreferenten verschickt werden können.
Rücksetzen	Im Falle der datenbankbasierten Persistenz (in der Konfigurationsdatei `web.xml` einstellbar) kann der Metadatensatz auf seinen ursprünglichen Stand zurückgesetzt werden, so dass wieder genau die Angaben des Autors enthalten sind.
Export DDB	Hier erfolgt die automatische Konvertierung des Metadatensatzes in das von der DDB erwartete Format und der eigentliche Export.
Export SWB	Hier erfolgt die automatische Konvertierung des Metadatensatzes in das von der SWB erwartete Format und der eigentliche Export.
Löschen	Löscht den selektierten Metadatensatz nach vorheriger Warnung. Der Metadatensatz ist damit endgültig nicht mehr verfügbar!
Einstellen	Beginnt mit der Einstellung des selektierten Metadatensatzes (ein mehrstufiger Prozess), falls dieser alle Voraussetzungen dafür erfüllt (siehe dazu auch Abschnitt 5.1).

Tabelle 9 *- Bearbeitungsmöglichkeiten von Metadatensätzen*

Versucht man, den in Abschnitt 3.1 spezifizierten Geschäftsablauf anhand der Teilanwendung *AdminTools* zu demonstrieren, erfolgte also nach einer erfolgreichen Anmeldung und der Sichtung aller aktuellen Metadatensätze die Selektion eines (angenommenen) neuen Metadatensatzes, der erst kurze Zeit zuvor von einem Autor erstellt wurde. Danach würde der Nutzer (ein Mitarbeiter des Referates 5.3) auf „Bearbeiten“ klicken, um sich die vorhandenen Metadaten (Primär-Metadaten) des Metadatensatzes anzeigen zu lassen. Diese werden von der Anwendung wie in **Abbildung 42** auf der folgenden Seite präsentiert. Die einzelnen *Dublin Core*-Kategorien des Metadatensatzes inklusive der temporären Metadaten (in der Abbildung nicht zu sehen) werden separat und übersichtlich angezeigt. Dabei werden die drei verschiedenen Arten von Metadaten[8] auch jeweils farblich unterschiedlich hervorgehoben, ebenso existiert eine farbliche Unterscheidung für obligatorische bzw. optionale Metadaten.

[8] die Metadaten eines Metadatensatzes unterteilen sich in:
-Primär-Metadaten: Metadaten des Autors
-Sekundär-Metadaten: Metadaten der Bibliothek (Klassifikationen etc.)
-System-Metadaten: vom System vorgegebene, in der Regel nicht änderbare Metadaten

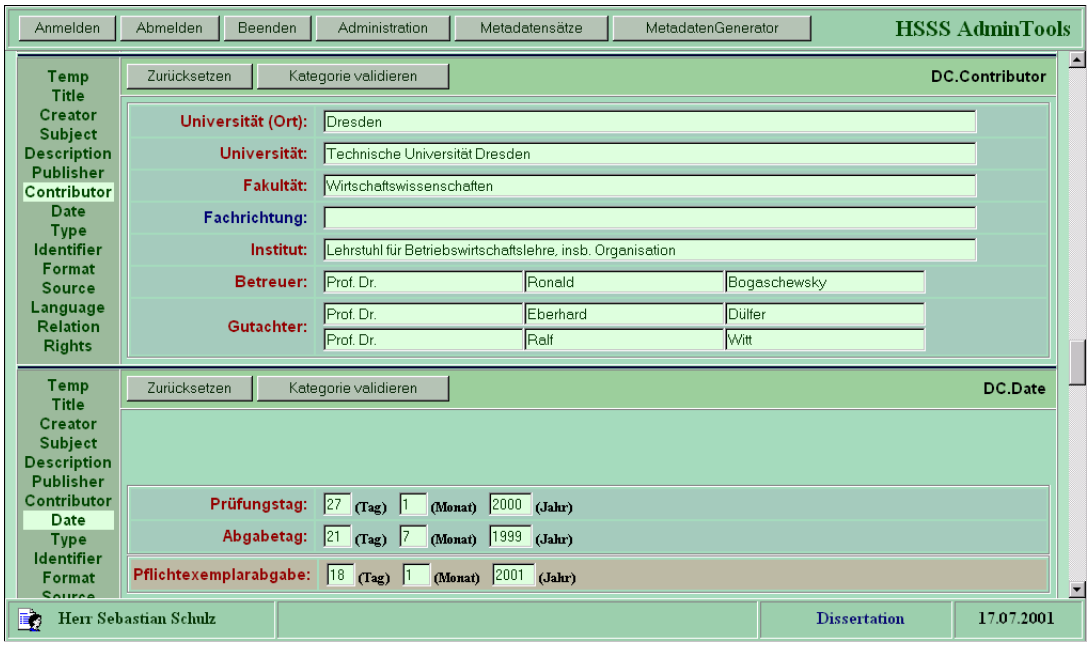

Abbildung 42 *- HSSS AdminTools: Die Metadaten eines Metadatensatzes*

Über die Navigationsleiste, die sich auf der linken Seite jeder *Dublin Core*-Kategorie befindet, kann man bequem zwischen den einzelnen Kategorien wechseln. Bis auf die System-Metadaten sind alle Metadaten editierbar, so dass vorhandene Metadaten (i.d.R. Primär-Metadaten) gegebenenfalls berichtigt und Sekundär-Metadaten hinzugefügt werden können. Im angenommenen Fall eines neuen Metadatensatzes würde der Nutzer die vorhanden primären Metadaten auf syntaktische und semantische Korrektheit überprüfen und anschließend auf den nicht in der Abbildung gezeigten Button „Änderungen speichern“ klicken. Auch wenn keine Veränderungen vorgenommen worden sind, werden alle Metadaten des entsprechenden Metadatensatzes in Abhängigkeit seines Metadaten-Zustandes von den *AdminTools* Server-seitig validiert. Bei einem neuen Metadatensatz, dessen Metadaten-Zustand dem Ausgangszustand „Validierung Primärmetadaten“ entspricht, würde die Anwendung den Metadaten-Zustand also je nach Ergebnis der Validierung auf „Validierung Primärmetadaten OK“ oder „Validierung Primärmetadaten Fehler“ (mit entsprechender) Fehlermeldung setzen. Im ersten Fall, wenn alle Primär-Metadaten korrekt validiert vorliegen, erfolgt der Schritt der Übermittlung der primären Metadaten an den jeweils zuständigen Fachreferenten. Dazu verlässt der Nutzer das hier beschriebene Bearbeitungsfenster, indem er auf “Metadatensätze“ in der oberen Navigationsleiste klickt. Er gelangt auf die anfangs beschriebene Seite mit der Metadatensatz-Übersicht, wo der gerade bearbeitete Metadatensatz der jeweiligen Dokumentenart[9] noch selektiert ist. Durch ein Klicken des

[9]in diesem Beispiel wird angenommen, dass es sich um die Dokumentenart „Dissertation“ handelt

Buttons „Drucken“ gelangt der Nutzer auf eine Seite, die Teile der primären Metadaten des ausgewählten Metadatensatzes in einer druckfreundlichen HTML-Präsentation zeigt, welche durch ein Klicken auf „Drucken“ ausgedruckt werden kann. Alternativ kann von der Anwendung auch ein entsprechendes PDF-Dokument zur Laufzeit aus den Metadaten erstellt werden[10], welches über ein entsprechendes Plugin wie in **Abbildung 43** gezeigt integriert wird:

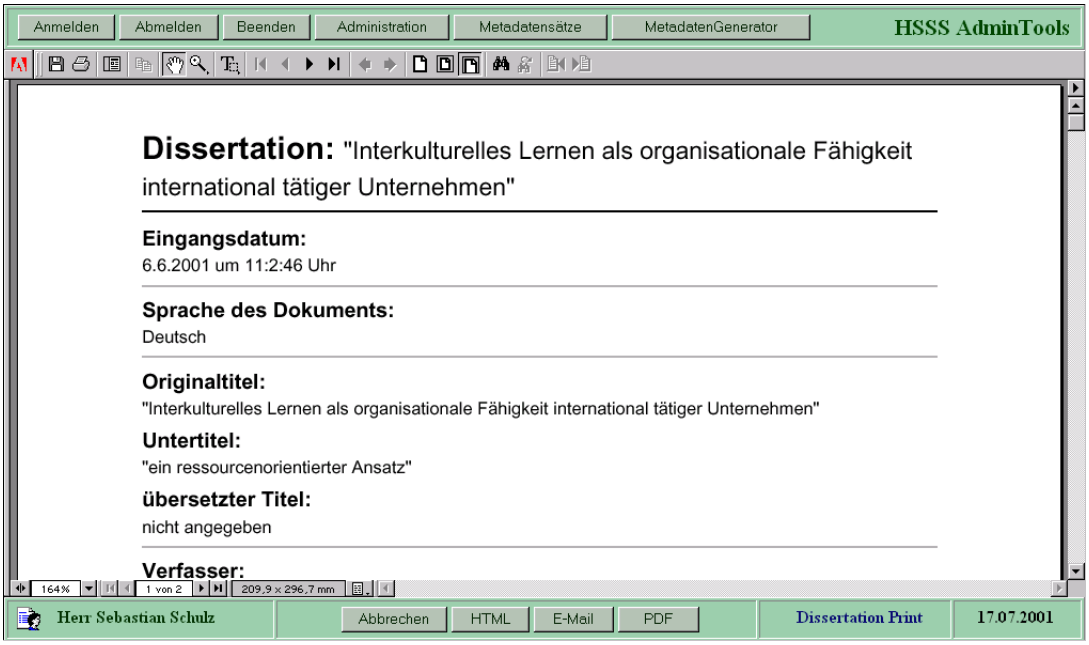

Abbildung 43 *- HSSS AdminTools: Druck-Präsentation der Primär-Metadaten*

Der Nutzer kann aber auch durch ein Klicken auf „E-Mail“ im unteren Teil der Abbildung automatisch ein von Web-basierten E-Mail-Diensten bekanntes Formular erstellen lassen, dass im Nachrichtenfeld bereits die primären Metadaten enthält. Es muss nur noch die Adresse des jeweiligen Fachreferenten angegeben werden, und durch ein weiteres Klicken auf „Absenden“ würde die Nachricht von der Anwendung automatisch an den Fachreferenten gesendet[11]. Im weiteren Verlauf des Geschäftsganges würde der Fachreferent auf der Grundlage des Druckexemplars, der elektronischen Version und den erhaltenen primären Metadaten die Sekundär-Metadaten erstellen. Diese sendet er entweder per E-Mail oder durch die Hauspost wieder an das Referat 5.3 zurück. Hier würde man den Geschäftsgang mit Hilfe der *AdminTools* fortsetzen, indem in dem Bearbeitungsbereich (siehe **Abbildung 42**) die entsprechenden

[10] dies geschieht mit Hilfe von XML-Technologien, insbesondere durch Xalan und FOP (siehe Anhang D)

[11] Zu diesem Zwecke wird das JavaMail API verwendet, so dass die Anwendung nicht auf externe (lokale) E-Mail-Clients angewiesen ist. Näheres siehe Abschnitt D im Anhang.

Sekundär-Metadaten des Fachreferenten für den Metadatensatz eingegeben werden. Diese Metadaten werden anschließend validiert, indem der Metadaten-Zustand zuerst auf „Validierung Sekundärmetadaten“ gesetzt und anschließend wieder auf „Änderungen speichern“ geklickt wird. Sind alle, insbesondere die sekundären Metadaten vollständig und korrekt, befindet sich der Metadatensatz im finalen Metadatenstatus „Validierung Sekundärmetadaten OK“. Der Metadatensatz ist also vollständig und kann damit bereits an die externen Institutionen SWB und DDB exportiert werden. Dazu klickt der Nutzer im zentralen Übersichtsbereich für den entsprechenden Metadatensatz auf “Export DDB“ bzw. “Export SWB“. Im ersten Fall erscheint ein Formular wie das in **Abbildung 44** gezeigte[12]:

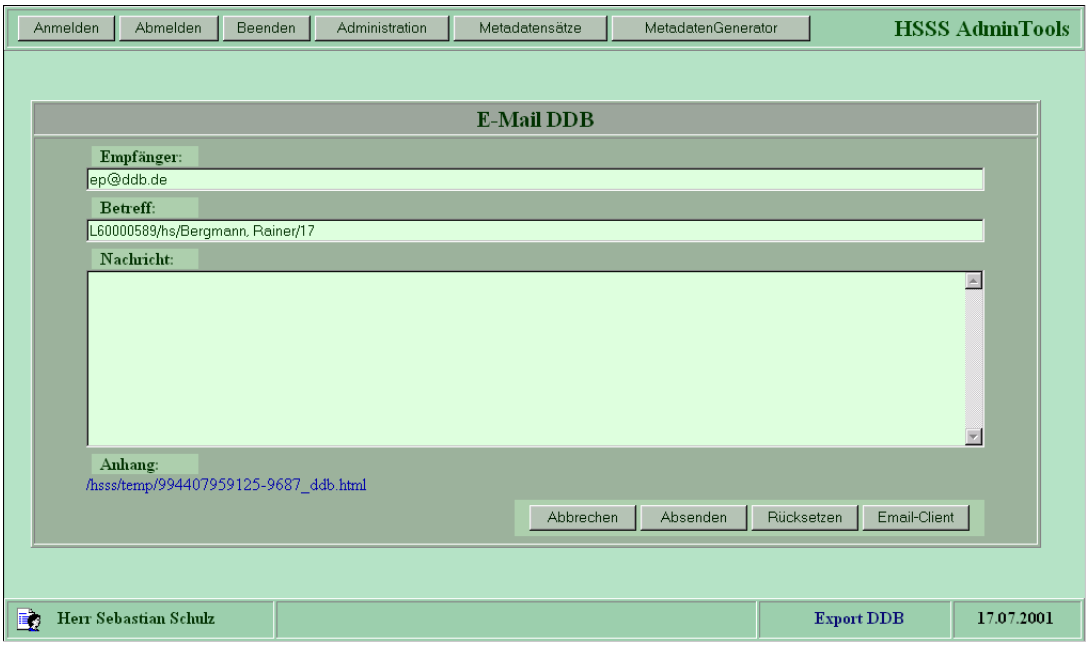

Abbildung 44 *- HSSS AdminTools: Export nach DDB*

Da der Metadatenexport nach DDB E-Mail-basiert verläuft, ist in der Abbildung ein typisches E-Mail-Formular zu sehen, welches von den *AdminTools* dynamisch erstellt wird. Die Zieladresse ist bereits eingetragen, sie wird aus der Konfigurationsstruktur `hsssdata.xml` gewonnen (siehe auch Abschnitt 5.1). Der Betreff wird ebenfalls dynamisch generiert und entspricht einem von der DDB vorgegebenen Format. Optional kann der Nutzer nun noch eine Nachricht hinzufügen. Die eigentlichen Export-Daten befinden sich aber in einer HTML-Datei, die der E-Mail als Anhang

[12]Würde diese Aktion für einen Metadatensatz getätigt werden, dessen Metadaten-Zustand noch nicht „Validierung Sekundärmetadaten OK“ entspricht, wäre der Button „Absenden“ gesperrt, weil ein Export in diesem Stadium auch noch keinen Sinn machen würde.

beizufügen ist. Diese Datei wurde von der Anwendung ebenfalls komplett automatisiert mit Hilfe von XSLT[L-XLa][13] erstellt und ist durch den Nutzer einsehbar, indem er auf den Link unter „Anhang“ klickt. Der Klick auf „Absenden“ sendet an die voreingestellte Adresse der DDB eine E-Mail mit dem beschriebenen Betreff, einem optionalen Nachrichtentext und einer HTML-basierten Anhangsdatei, in der die Exportmetadaten in dem vorgeschriebenen Format der DDB kodiert sind.

Im Falle des SWB-Exportes wird ein ähnliches Fenster präsentiert, allerdings erfolgt der Export FTP-basiert, so dass zur Realisierung natürlich andere Technologien verwendet werden[14]. Die Adresse des FTP-Servers ist der Anwendung ebenfalls bereits durch die Konfigurationsstruktur `hsssdata.xml` bekannt. Auch in diesem Fall wird zur Kodierung der Export-Informationen eine HTML-basierte Datei verwendet, die ebenfalls automatisch erstellt wird und vom Nutzer eingesehen werden kann. Bei einem Klick auf den Button „in FTP-Verzeichnis einstellen“ wird die Datei automatisch auf dem spezifizierten Server in das angegebene Verzeichnis abgelegt. Für beide Exporte wird der Exportstatus von der Anwendung automatisch auf „Metadaten exportiert“ gesetzt, gleichzeitig wird der genaue Zeitpunkt des Exports vermerkt.

Nun sind fast alle Voraussetzungen für das Einstellen der Metadaten in die persistente Datenbasis des Gesamtsystems gegeben. Parallel zu den bisher beschriebenen Schritten sollte ebenfalls das Dokument bearbeitet worden sein. Die Bearbeitung besteht am Anfang aus vielen manuellen Schritten, die nicht automatisierbar sind und den *AdminTools* durch das korrekte Setzen des Dokumenten-Zustands in dem dafür vorgesehenen Bereich der Teilanwendung (über den Button „Dokument“) mitgeteilt werden müssen. Zu diesen Bearbeitungsschritten gehört das formale Vergleichen des elektronischen Dokuments mit der Druckversion, was bei positivem Verlauf durch die Dokumenten-Zustandsänderung auf “Validierung Dokument OK“ mitgeteilt werden sollte, und das Konvertieren in das HSSS-Präsentationsformat, welches zu dem Dokumentenstatus “Konvertierung Dokument OK“ führen sollte. Diese Schritte geschehen aber vollständig außerhalb der Anwendung zum Beispiel auf dem lokalen Rechner eines Mitarbeiters des Referats 5.3. Sind sowohl Validierung als auch Konvertierung des Dokuments abgeschlossen, benötigt die Anwendung Zugriff auf das Dokument. Zu diesem Zwecke existiert ein freigegebenes Verzeichnis auf dem Server, auf dem die Anwendung temporäre Dokumente erwartet. Diese Verzeichnis ist bei den Mitarbeitern des Referats 5.3 als entferntes Laufwerk angebunden, so dass sie Dokumente entsprechend einstellen können. Dabei ist es möglich, innerhalb des vorgesehenen Verzeichnisses weitere Unterverzeichnisse zu schaffen und somit eine strukturierte Ablage für verschiedene Hochschulschriften zu schaffen. Dieses Verzeichnis muss der Anwendung

[13] und unter Verwendung des XSLT-Prozessors Xalan der ASF, siehe Anhand D

[14] In diesem Fall das API „finj“, welches wieder näher im Anhang D beschrieben ist

mitgeteilt werden, was durch das Selektieren des Metadatensatzes und anschließendes Klicken des Buttons „Dokument“ vorbereitet wird. Hier können das entsprechende Verzeichnis und das Dokument ausgewählt werden. Anschließend kann der Dokumentenzustand auf den finalen Zustand „Dokument OK“ gesetzt werden. Dadurch wird der Gesamtzustand des Metadatensatzes von der Anwendung automatisch auf „Vorgang abgeschlossen“ gesetzt, da alle dafür erforderlichen Voraussetzungen erfüllt sind (siehe dazu auch Abschnitt 5.1). **Abbildung 45** zeigt einen Ausschnitt der temporären Metadaten, wie ihn der entsprechende Bereich der Anwendung jetzt darstellen würde:

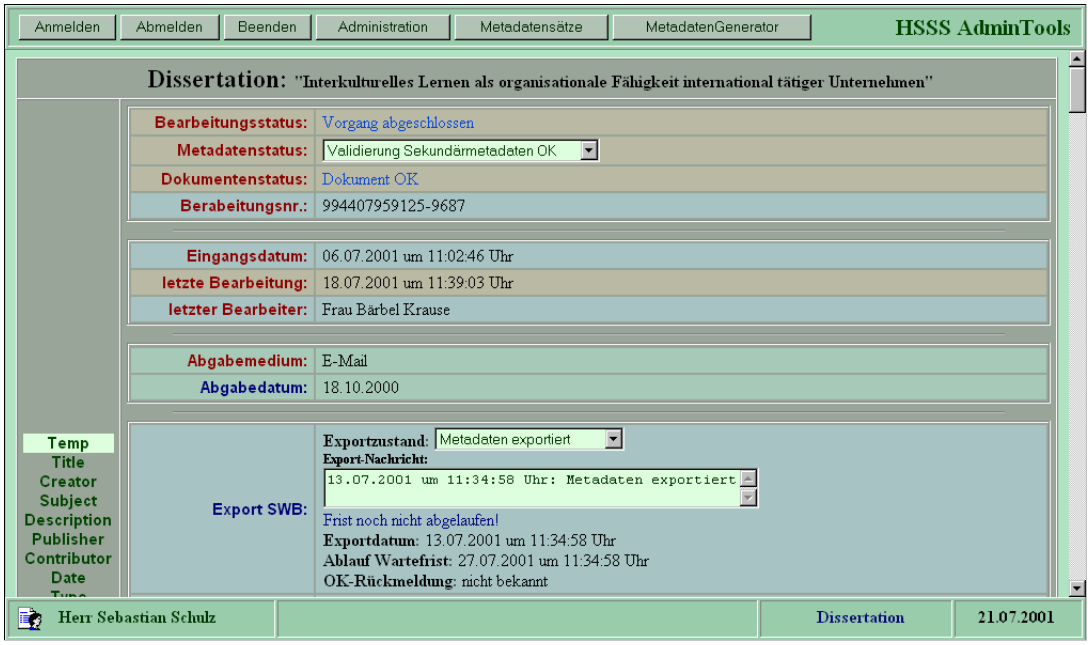

Abbildung 45 *- HSSS AdminTools: temporäre Metadaten, Zustände*

Wie zu erkennen ist, entsprechen alle Teilzustände dem jeweiligen erfolgreichen Endzustand. Die Abbildung verdeutlicht ebenfalls noch einmal die schon angesprochene separate Zustandsbeschreibung für die Exporte, hier anhand der Informationen für den Export nach SWB. Der Exportzustand entspricht „Metadaten exportiert“ und der genaue Export-Zeitpunkt wurde von der Anwendung festgehalten. Zusätzlich wird der Zeitpunkt des Ablaufs der Wartefrist berechnet und angezeigt. Der Nutzer hat außerdem die Möglichkeit, der Exportnachricht eigene Mitteilungen hinzuzufügen, etwa bei besonderen Vorkommnissen als Information für andere Bearbeiter. Ein ähnlicher Formularbereich innerhalb der temporären Metadaten existiert selbstverständlich auch für den Metadatenexport nach DDB.

Der Metadatensatz kann nun endgültig eingestellt werden. Dabei handelt es sich um einen mehrstufigen Vorgang, der dadurch eingeleitet wird, dass für den betrachteten Metadatensatz die Aktion „Einstellen“ gewählt wird. Anschließend erfolgt ein Hinweis, dass diese Operation auf die Sichtbarkeit des Metadatensatzes sowohl innerhalb der *AdminTools* als auch von außen große Auswirkungen hat. Falls die Rückmeldungen für die Exporte noch nicht vorliegen und die Wartefrist noch nicht vorüber ist, gibt die Anwendung außerdem entsprechende Warnmeldungen aus. Der Vorgang des Einstellens kann jedoch trotzdem fortgesetzt werden und würde als nächstes zu der in **Abbildung 46** gezeigten Ansicht führen:

Anmelden
Abmelden
Beenden
Administration
Metadatensätze
MetadatenGenerator
HSSS AdminTools
Einstellen der Metadaten
Dokument-Bearbeitung
Das PDF-Dokument der Hochschulschrift kann jetzt aus dem temporären Verzeichniss
http://hsss.slub-dresden.de/tempdocroot/diss_buenning.pdf
eingelesen und bearbeitet werden (Text-Extraktion, Hinzufügen der HSSS-Seite).
Sie sollten die automatische Text-Extraktion auf jeden Fall einmal ausführen! Falls Sie allerdings nicht erfolgreich war, und sie eine Text-Datei von Hand konvertiert und erstellt haben, müssen Sie Überspringen wählen, damit Ihr Ergebnis nicht überschrieben wird.
Soll Die automatische Volltext-Erstellung sowie die Dokumentmarkierung jetz vorgenommen werden?
Dieser Vorgang kann einige Zeit (bis zu 3 Minuten) in Anspruch nehmen. (bitte haben Sie etwas Geduld)
[Ausführen = Ja, Überspringen = Nein]
<< Zurück
Überspringen
Ausführen
Herr Sebastian Schulz
Dokument bearbeiten
Dissertation Einstellen
17.07.2001

Abbildung 46 *- HSSS AdminTools: Einstellen des Metadatensatzes*

In dieser Abbildung macht die separate Navigationsleiste noch einmal deutlich, dass es sich bei dem Einstellen um einen aus mehreren sequentiellen Teilvorgängen bestehenden Vorgang handelt, der auch jederzeit durch ein Klicken auf „Abbrechen“ abgebrochen werden kann bzw. der es erlaubt, bestimmte Teilvorgänge auch zu überspringen. Außerdem kann über den Button „Zurück“ jederzeit wieder zum vorangegangenen Teilvorgang zurückgegangen werden. Zunächst würde die Anwendung bei einem Klicken auf „Ausführen“ versuchen, aus dem elektronischen Dokument im Präsentationsformat automatisch den Volltext zu extrahieren. Da das Ergebnis von unterschiedlicher Qualität sein kann, wird es in einem folgenden Fenster dem Nutzer präsentiert. So kann dieser entscheiden, ob der erstellte Volltext den bestehenden Ansprüchen genügt. In diesem Fall würde er den Vorgang des Einstellens fortsetzen. Im anderen Fall besteht die Möglichkeit, den Vorgang vorerst abzubrechen, um

zum Beispiel durch die Verwendung anderer Werkzeuge außerhalb der Anwendung zu einem besseren Ergebnis zu gelangen. Der nächste Teilschritt besteht im Kopieren des Dokuments in ein entsprechendes Verzeichnis auf dem Webserver und kann auch wieder wahlweise übersprungen werden. Der jeweilige Verzeichnispfad wird aus den Metadaten automatisch in einer bestimmten Form relativ zum Wurzelverzeichnis des Webservers generiert. Für das betrachtete Beispiel würde der Verzeichnispfad

```
<webserver-root/dissertation/2000/wirtschaftswissenschaften/994407959125-9687/
```

lauten. Das Verzeichnis setzt sich also aus der Dokumentenart, dem Veröffentlichungsjahr, der Fakultät und der Bearbeitungsnummer des jeweiligen Metadatensatzes zusammen. In dieses Verzeichnis wird das Dokument unter dem Namen „`<Bearbeitungsnummer>.<Erweiterung des Präsentationsformats>`“ kopiert[15]. Gleichzeitig wird das Dokument in ein ähnlich strukturiertes Verzeichnis außerhalb des Webservers kopiert. Dabei handelt es sich um eine Art „Spiegelverzeichnis“, welches dazu dient, neben dem Präsentationsformat auch alle anderen von der Hochschulschrift vorhandenen Formate zu archivieren. Dieses Verzeichnis wird auch im Rahmen einer Backup-Strategie regelmäßig gesichert. Nach diesem Teilschritt wird ein weiteres Formular präsentiert, welches darauf hinweist, dass der Einstellvorgang jetzt beendet wird. Mit der Ausführung der nächsten Schritte, die innerhalb einer Transaktion ablaufen, ist das Einstellen endgültig und demzufolge auch nicht wieder rückgängig zu machen[16]. Bei diesen transaktionalen Schritten handelt es sich um die im Folgenden aufgezählten:

- Der Metadatensatz (mit Ausnahme der temporären Metadaten) wird unter Zuhilfenahme entsprechender Klassen und einiger gespeicherter Prozeduren auf der Datenbankseite in die Datenbasis eingestellt (siehe dazu auch Abschnitt 5.3). Dazu wird vorher durch eine spezielle gespeicherte Prozedur geprüft, ob der Metadatensatz schon existiert und dieser gegebenenfalls gelöscht. Dies ist notwendig, weil die Teilanwendung *AdminTools* ebenfalls ermöglicht, bereits eingestellte Metadatensätze zu bearbeiten und anschließend erneut einzustellen.

- Anhand der vorhandenen Metadaten werden in einer bestimmten Tabelle (`stat_metadatasets`, siehe auch Abschnitt C im Anhang) statistische Informationen für den Metadatensatz hinzugefügt.

- Der Metadatensatz wird in die Tabelle der eingestellten temporären Metadatensätze geschrieben(`temp_metadatasets_finished`, siehe auch Abschnitt C im Anhang).

[15] im betrachteten Beispiel also `994407959125-9687.pdf`
[16] zumindest nicht mit den Mitteln der *AdminTools*

- Der Metadatensatz wird aus der Tabelle der aktuellen temporären Metadatensätze gelöscht(`temp_metadatasets`, siehe auch Abschnitt C im Anhang).
- Es erfolgt eine automatische Erstellung der Frontdoor unter Zuhilfenahme eines XSL-Stylesheets, welches aus der XML-basierten Metadatensatzstruktur die vorgesehene XHTML-Datei erstellt und in das vorher gebildete Webserver-Verzeichnis einstellt, in dem sich auch bereits das Dokument im Präsentationsformat befindet.

Alle diese hier beschriebenen Aktionen werden entweder gemeinsam und korrekt ausgeführt oder zurückgesetzt[17]. Konnte die Transaktion erfolgreich abgeschlossen werden, sind eine Reihe weiterer Aktionen erforderlich, welche in der folgenden Aufzählung aufgelistet sind:

- Die Abbildung der statischen URL's auf reale URL's, die in Abschnitt 6.1 ab Seite 178 näher beschrieben wird, muss für den aktuellen Metadatensatz aktualisiert werden.
- Der Index der verwendeten Volltextsuchmaschine, die in Abschnitt 6.2 ab Seite 185 näher erläutert wird, muss für den aktuellen Metadatensatz inkrementell aktualisiert werden.
- Für den Fall einer erstmaligen Einstellung des Metadatensatzes wird an den Autor eine E-Mail versendet, die ihn darauf hinweist, dass seine Hochschulschrift erfolgreich persistent archiviert wurde und die ihm die absolute URL zur Frontdoor seiner Hochschulschrift mitteilt, die er im Folgenden als Referenz benutzen kann. Gleichzeitig ist er aufgefordert, die von der Anwendung erstellte Präsentation der Metadaten (die Frontdoor) und die elektronische Version seines Dokuments im Präsentationsformat noch einmal auf inhaltliche Korrektheit und Vollständigkeit zu überprüfen.
- Der Metadatensatz wird als Objekt aus der Objektstruktur aller temporären Metadatensätze im Hauptspeicher der Anwendung gelöscht, so dass er auch in dem Übersichtsformular in dieser Rubrik nicht mehr erscheint.

Wie im letzten Punkt angedeutet, erscheint der Metadatensatz in der in **Abbildung 41** gezeigten Metadatensatzübersicht nicht mehr. Stattdessen ist dieser Metadatensatz in der Kategorie der eingestellten Metadatensätze sichtbar, die man durch das

[17] Wobei die Erstellung der Frontdoor nicht automatisch zurückgesetzt werden kann (da es keine Datenbankoperation ist), sondern explizit durch die Anwendung zurückgesetzt (gelöscht) werden muss.

Ändern des Radiobuttons von „aktuelle“ auf „eingestellte“ und anschließendes Klicken auf „Aktualisieren“ erreicht. Dort findet sich eine ähnliche Auflistung, wobei der Zustand eines jeden Metadatensatzes im Normalfall natürlich „Vorgang abgeschlossen“ entsprechen sollte. Für die einzelnen Metadatensätze sind dieselben Aktionen möglich, so dass auch im Nachhinein noch Metadaten („Bearbeiten“) und Dokumentinformationen („Dokument“) verändert werden können. Auch das Drucken von Primärmetadaten („Drucken“) ist möglich und die Metadatenexporte können gegebenenfalls noch einmal ausgeführt werden („Export DDB“ bzw. „Export SWB“). Das Rücksetzen des Metadatensatzes („Rücksetzen“) führt auf die zuletzt eingestellte Version zurück. Das Löschen des Metadatensatzes durch “Löschen“ würde den Metadatensatz zwar aus der Tabelle `temp_metadatesets_finished`, nicht jedoch aus der persistenten Datenbasis des Systems[18] entfernen. „Einstellen“ würde einen bereits eingestellten Metadatensatz noch einmal einstellen (also aktualisieren), wobei alle oben beschriebenen Schritte ausgeführt werden[19]. So können auch im Nachhinein noch Änderungen an den Metadatensätzen in der persistenten Datenbasis des Gesamtsystems durch die Teilanwendung *AdminTools* vorgenommen werden.

5.2.2. Administrationsunterstützung

Neben den im vorigen Abschnitt vorgestellten Funktionalitäten, die die Geschäftsmodelle der Dokumentenarten technisch begleiten, bieten die *AdminTools* eine Reihe zusätzlicher Funktionalitäten, welche die Administration des Gesamtsystems unterstützen. Um diese Möglichkeiten nutzen zu können, klickt ein Nutzer in der oberen Navigationsleiste den Button „Administration“. Dadurch wird unter anderem die obere Navigationsleiste ausgewechselt und mit neuen, sich auf Administrationsfunktionalitäten beziehenden Buttons bestückt. **Abbildung 47** zeigt diese Navigationsleiste, gleichzeitig wurde hier bereits die Aktion „Benutzerverwaltung“ gewählt, welche die Anwendung zur dynamischen Erzeugung des auf der folgenden Seite abgebildeten Formulars veranlasst.

[18] Damit sind die eigentlichen Persistenz-Tabellen (`pers_*`) innerhalb der Datenbank gemeint (siehe auch Abschnitt C im Anhang). Bei der Tabelle `temp_metadatsets_finished` handelt es sich quasi um eine vorgelagerte „Stellvertreter“-Tabelle für die *AdminTools*, welche dieser Teilanwendung eine für sie optimierte Sicht und einen direkteren Zugriff auf die entsprechenden Metadatenstrukturen bietet.

[19] mit Ausnahme der E-Mail-Benachrichtigung des Autors

Abbildung 47 *- HSSS AdminTools: Benutzerverwaltung*

Wie zu erkennen ist, bestehen die möglichen Administrationsaktionen durch das Klicken der Buttons „Benutzerverwaltung“, „Konfiguration“, „Stylesheets“, „Logdateien“, „SQL“ bzw. „ToolBox“, wobei in der Abbildung bereits die Aktion „Benutzerverwaltung“ gewählt wurde. Der Button „Hauptseite“ führt den Nutzer wieder auf die schon bekannte, in **Abbildung 41** dargestellte Übersichtsseite zurück. Wie dem abgebildeten Formular zur Benutzerverwaltung zu entnehmen ist, unterscheidet die Anwendung zwei Arten von Nutzern: „normale“ Nutzer und solche mit Administrations-Privilegien (wie im betrachteten Fall). Nutzer mit Administrationsrechten können neben dem eigenen Account auch andere Accounts ändern und löschen bzw. neue Accounts erstellen. Sie sind zusätzlich durch ein Administrationssymbol in der linken unteren Ecke neben dem Benutzernamen gekennzeichnet. Gewöhnliche Nutzer haben im Unterschied zu Administrations-Benutzern zu bestimmten Administrationsformularen keinen Zutritt, so zum Beispiel auch für das in der **Abbildung 48** dargestellte, durch die Aktion „Konfiguration“ erreichbare Formular. Hier können alle in Abschnitt 3.3.1.2 bzw. B beschriebenen Konfigurationsstrukturen angezeigt, bearbeitet und zurückgespeichert werden, unabhängig davon, ob diese dateibasiert oder datenbankbasiert verwendet werden. Eine Änderung von `web.xml` würde aber erst nach einem Neustart des Servlet-Containers wirksam werden, was von den *AdminTools* aus durch die Aktion “Start&Stop“ möglich ist. Die anderen Konfigurationsstrukturen werden in der Regel beim nächsten Zugriff eines Clients wirksam und können durch

die Aktion „Reload“ sogar in die aktuellen Sitzungen übernommen werden[20]. Die XML-basierten Konfigurationsstrukturen müssen von den berechtigten Nutzern mit Administrationsprivilegien mit großer Vorsicht geändert werden, da sich hier weite Teile des Gesamtsystems beeinflussen lassen und eine korrupte Konfigurationsstruktur unter Umständen das gesamte System vorübergehend unbenutzbar macht. In diesem Fall muss die betroffene Konfigurationsstruktur repariert oder durch eine Sicherung ersetzt werden. Bei richtigem Gebrauch kommen allerdings die Vorteile der Architektur an dieser Stelle voll zum Tragen, da weite Teile der Anwendung und deren Verhalten geändert werden können, ohne eine einzige Zeile Quellcode zu ändern. Natürlich wäre eine Präsentation der jeweiligen Konfigurationsstruktur innerhalb eines Formulars ein nächster Schritt in Richtung Nutzerfreundlichkeit und Transparenz, da er die XML-Strukturen verstecken würde. Eine solche Funktionalität konnte aber aufgrund des hohen Aufwands im Rahmen der Diplomarbeit nicht realisiert werden und bleibt einer späteren Ausbaustufe des HSSS vorbehalten.

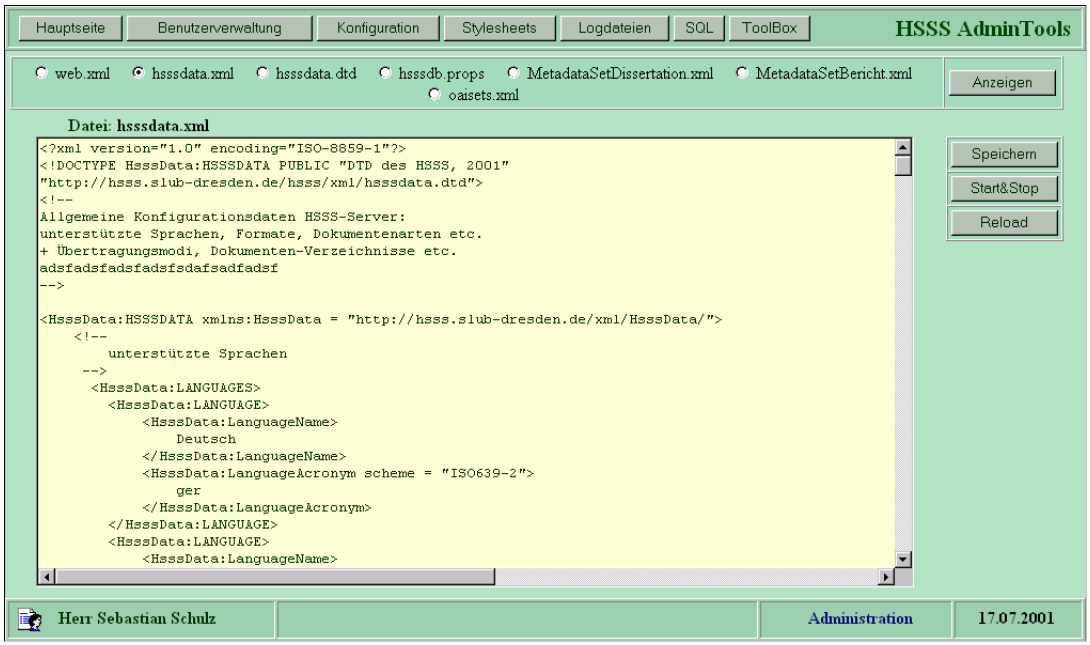

Abbildung 48 *- HSSS AdminTools: Konfigurationsstrukturen*

Durch die Aktion „Stylesheets“ wird ein der vorigen Abbildung ähnliches Formular präsentiert, welches ebenfalls nur Administrations-Nutzern zugänglich ist. Hier können alle in der Anwendung genutzten XSL-Stylesheets angezeigt, geändert und gespeichert

[20]So würde beispielsweise die Erweiterung der unterstützten Sprachen in der Konfigurationsstruktur`hsssdata.xml` um eine neue Sprache nach der Aktivierung von „Reload“ auch in bereits parallel laufenden Instanzen des *MetadatenGenerators* sichtbar.

werden. Dies hat den großen Vorteil, dass zur Laufzeit zahlreiche Änderungen möglich sind, die ansonsten Programmänderungen, ein erneutes Kompilieren und Deployment sowie das Neustarten der beteiligten Systemkomponenten erfordern würden. So lassen sich Änderungen im Hinblick auf neue Anforderungen an die Export-Formate für DDB (*Die Deutsche Bibliothek*) und SWB (*Südwestdeutscher Bibliotheksverbund*) durchführen, außerdem lassen sich die verschiedenen Ansichten der Primärmetadaten für den Fachreferenten (HTML-Version, Text-Version für die E-Mail, PDF-Version) auf diese Art und Weise ändern. Schließlich kann auch die Vorlage zur automatischen Erstellung der Frontdoors an dieser Stelle erweitert bzw. geändert werden.

Durch das Klicken des Buttons „Logdateien“ wird dem Nutzer ein Formular präsentiert, auf dem er sich alle Logdateien des Systems anzeigen lassen kann, was insbesondere der Fehlersuche dient. „SQL“ aktiviert einen Web-basierten SQL-Client, durch den SQL-Befehle direkt an die Datenbank des HSSS abgesetzt werden können. Der Button „ToolBox“ führt zu einer Sammlung von wichtigen Aktionen zur Administration des Gesamtsystems. Auch dieses Formular ist nur Administrator-Nutzern zugänglich und bietet zur Zeit die Aktionen „Frontdoors aktualisieren“ und „Index neu aufbauen“. Erstere Aktion erstellt für alle Metadatensätze, die in der persistenten Datenbasis gespeichert sind, die Frontdoor neu. Diese Aktion ist insbesondere dann sinnvoll, wenn durch die Aktion „Stylesheets“ die Vorlage zur automatischen Generierung der Frontdoors geändert wurde und sich diese Änderung nicht nur auf alle noch folgenden Metadatensätze, sondern auch auf die schon eingestellten auswirken soll. Die zweite Aktion „Index neu aufbauen“ führt dazu, dass der Index der verwendeten Volltextsuchmaschine (siehe dazu Abschnitt 6.2 ab Seite 185) komplett neu für alle persistent eingestellten Metadatensätze erstellt wird.

5.3. Persistenzrealisierung

Nachdem im Abschnitt 4.3 auf Seite 134 bereits die zu diesem Zeitpunkt genutzten Teile des Datenbank-Schemas vorgestellt wurden, welche allerdings nur sehr flach strukturiert waren, wird in diesem Abschnitt das Design des „eigentlichen“, für die eingestellten Metadatensätze verwendeten Datenbank-Schemas vorgestellt. Das zu spezifizierende Schema soll dabei alle gemeinhin „üblichen“ Datenbankschema-Eigenschaften[HeS00] erreichen, insbesondere sollen

1. alle Anwendungsdaten aus den Basisrelationen hergeleitet werden können,
2. nur semantisch sinnvolle und konsistente Anwendungsdaten dargestellt werden können,
3. die Anwendungsdaten möglichst nicht-redundant dargestellt werden.

Aus dem ersten Punkt ergibt sich im vorliegenden Fall die Forderung, dass das Schema alle zu einem Metadatensatz gehörenden Metadaten[21] berücksichtigt und diese somit auch wieder hergeleitet werden können. Das bedeutet, dass alle in der Metadatenspezifikation der jeweiligen Dokumentenart definierten Elemente bzw. Subelemente in der Datenbank in einer noch zu bestimmenden Form gespeichert werden. Der zweite Punkt, der semantisch sinnvolle und konsistente Anwendungsdaten verlangt, wird zum Teil bereits durch die Arbeitsabläufe realisiert, deren Ziel es ja gerade ist, hochwertige, syntaktisch und semantisch korrekte Metadaten zu erstellen. Zu einem anderen Teil steht er in einem engen Zusammenhang mit dem Datenbankschema, da dieses durch ein bestimmtes Design (Normalisierungen, referentielle Integrität) konsistente Daten sogar bis zu einem bestimmten Grad erzwingen kann. Der dritte Punkt geht über den zweiten noch hinaus, indem durch die Forderung nach nicht-redundanten Strukturen bestimmte Anomalien[22] verhindert werden sollen.

Aus den drei beschriebenen Anforderungen ergibt sich als entscheidendes Entwurfsprinzip des Datenbankschemas die weitestgehende Normalisierung der Relationen, um funktionale Abhängigkeiten und damit mögliche Anomalien zu vermeiden. Ein erster

[21] Allerdings mit Ausnahme der temporären Metadaten, die nur während der Bearbeitung des Metadatensatzes erforderlich sind und die nicht in die persistente Datenbasis des Gesamtsystems übernommen werden sollen.

[22] Hier spielen insbesondere Update-Anomalien eine entscheidende Rolle

Schritt besteht dabei in der Unterstützung der ersten Normalform, die lediglich atomare Attribute in den Relationen erlaubt[23]. Auch die zweite Normalform soll durch das Datenbankschema vollständig unterstützt werden, so dass keine partiellen Abhängigkeiten zwischen Schlüsseln des Relationenschemas und weiteren Attributen bestehen. Auch die dritte Normalform, die transitive Abhängigkeiten beseitigt, wird von dem Datenbankschema unterstützt, wenngleich es an dieser Stelle nicht formal bewiesen wird. Durch die Unterstützung dieser Forderungen durch das Datenbankschema kann das Gesamtsystem bei der Schaffung und Erhaltung konsistenter, stabiler und nichtredundanter Anwendungsdaten optimal unterstützt werden, obwohl dieser Vorteil mit der Partitionierung der Daten über relativ viele Relationen erkauft wird. Die Partitionierung hält sich allerdings im Rahmen und soll durch die Forderung nach Minimalität des Datenbankschemas so weit es möglich ist begrenzt werden.

Im Folgenden soll das Datenbankschema anhand von drei hauptsächlich auftretenden Strukturen näher erläutert werden, wobei allerdings nicht jede Relation einzeln betrachtet werden kann. Eine Übersicht aller Relationen bzw. Tabellen sowie eine Auflistung aller Attribute und Besonderheiten der jeweiligen Tabellen befindet sich im Anhang C. Die Zentrale Relation des Datenbankschemas beinhaltet Informationen, die sich einem Metadatensatz eindeutig zuordenen lassen. Diese Relation ist in jeder der drei nachfolgend betrachteten Strukturen enthalten und und ist folgendermaßen aufgebaut:

`pers_metadataset(id, processingid, identifier_virtual, identifier_real, language_id, type_id, publisher_id, contributor_id, rights_id)`

Aus diesem Aufbau wird die erste der innerhalb des Schemas häufig auftretenden Strukturen bereits deutlich. Es handelt sich hierbei um 1:1-Beziehungen des Metadatensatzes zu anderen Relationen, wie formal in **Abbildung 49** dargestellt:

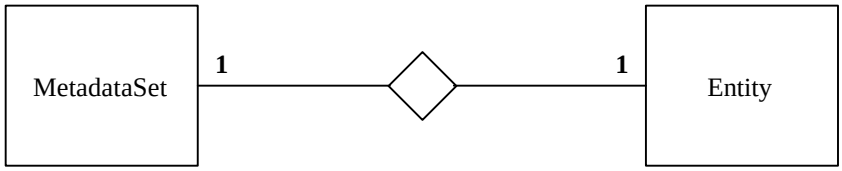

Abbildung 49 *- Grundlegende Schemastruktur 1*

Die Realisierung erfolgt dabei auf zwei verschiedene Arten: zum einen direkt innerhalb der `pers_metadataset`-Relation für pro Metadatensatz eindeutige Attribute wie

[23]Dies galt zum Beispiel für die in Abschnitt 4.3 betrachteten Relationen nicht, die für die ständige Bearbeitung bereits strukturierter Daten optimiert waren und als Attribute daher auch komplexe XML-Strukturen zuließen, die ihrerseits wieder aus hierarchisch strukturierten atomaren Attributen bestanden.

etwa der Bearbeitungsnummer (`processingid`), der absoluten URL (`identifier_virtuel`) oder der realen URL (`identifier_real`), zum anderen über Fremdschlüssel, die zu seperaten Relationen führen für nicht pro Metadatensatz eindeutige Attribute. Dazu zählen z. Bsp. die Sprache der Hochschulschrift (*`language_id`*), der Dokumententyp (*`type_id`*) oder die rechtlichen Bedingungen (*`rights_id`*). Diese wurden deshalb ausgelagert, um Redundanzen innerhalb der Relation `pers_metadatset` vermeiden zu können. Das Attribut `id` ist der Primärschlüssel der Relation und bei der Definition als *identity* gekennzeichnet worden, was den Datenbankserver dazu veranlasst, für einen innerhalb der Relation eindeutigen Wert zu sorgen[24]. Zwar existieren in dieser Relation mit `processingid` und auch `identifier_virtual` bzw. `identifier_real` noch andere Schlüsselkandidaten, allerdings ermöglicht ein automatisch inkrementierter, numerischer Wert ein noch besseres Indizieren durch die Datenbank (sowohl für Primärschlüssel als auch für Fremdschlüssel). Da fast alle Relationen direkt oder indirekt mit der Relation `pers_metadataset` zusammenhängen, wurde eine solche Identifikationsnummer für jede Relation des Schemas eingeführt. Als zweite grundsätzliche Struktur innerhalb des Datenbankschemas existiert die direkte N:M-Beziehung des Metadatensatzes zu anderen Entitäten, wie **Abbildung 50** verdeutlicht:

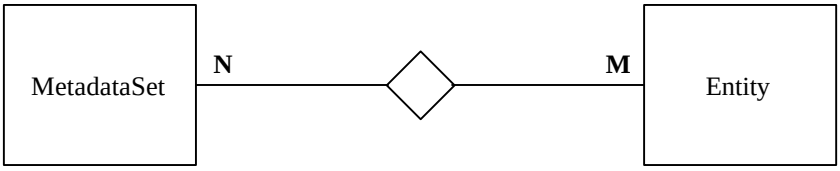

Abbildung 50 *- Grundlegende Schemastruktur 2*

Diese Struktur wird beispielsweise von den Publikationen (Veröffentlichungen) des Metadatensatzes realisiert. Diese Struktur und die entsprechend dazugehörigen Relationen werden nachfolgend am Beispiel der Publikationen einer Hochschulschrift demonstriert und sind im Zusammenhang mit der bereits gezeigten Relation `pers_metadataset` zu betrachten:

`pers_publication(id, name, isbn, annotation)`
`pers_publication_metadataset(id, publication_id, metadataset_id)`

Die Relation `pers_description` enthält alle Beschreibungen zu Hochschulschriften, die in verschiedenen Sprachen vorliegen können. Die Relation `pers_description_metadataset` ist eine reine Beziehungsrelation, die lediglich aus den Fremdschlüsseln der beiden beteiligten Relationen `pers_description` und `pers_metadataset` und dem

[24] Zusätzlich ist natürlich auch ein Datentyp anzugeben, in diesem Fall ist es `numeric(6)`, also eine 6-stellige Zahl. Die Eindeutigkeit wird durch das Inkrementieren dieses Attributs erreicht, weshalb auch die Bezeichnung „auto increment“ für Attribute dieses Typs gebräuchlich ist.

bereits erläuterten Primärschlüssel id[25] besteht. Als letzte hier betrachtete Struktur existiert innerhalb des Datenbankschemas die in **Abbildung 51** gezeigte:

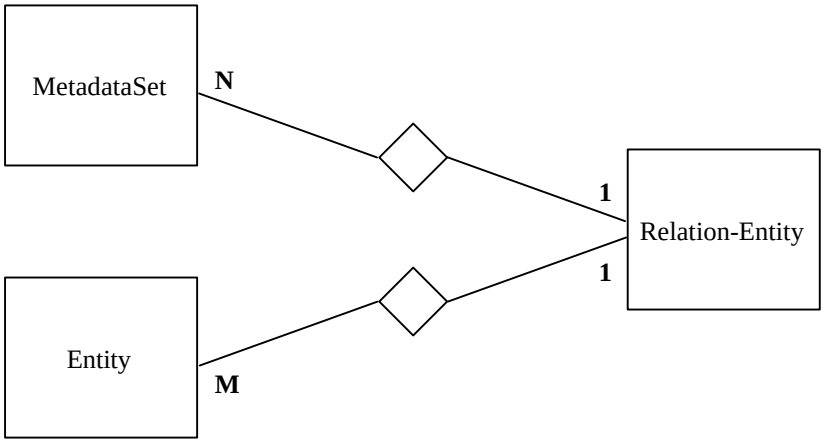

Abbildung 51 *- Grundlegende Schemastruktur 3*

Bei dieser Struktur handelt es sich um eine indirekte N:M-Beziehung zwischen Metadatensätzen und anderen Entitäten, welche durch ein weiteres Attribut der Beziehungstabelle näher bestimmt wird. Die im folgenden abgebildeten Relationen sollen die Struktur am Beispiel von Schlagworten bzw. Klassifikationen im Zusammenhang mit Metadatensätzen konkretisieren:

```
pers_subject(id, subject)
pers_subject_metadataset(id, type, subject_id, metadataset_id)
```

In der Relation `pers_subject` werden im Attribut `subject` alle Schlagworte und Klassifikationen für Hochschulschriften gespeichert. Da das Attribut als Schlüsselkandidat definiert ist, können keine Dopplungen auftreten. Die Beziehungsrelation `pers_subject_metadataset` ordnet Metadatensätzen Schlagworte bzw. Klassifikationen zu. Dabei bestimmt das Attribut `type` den Typ dieser Zuordnung und damit auch den Typ des Schlagwortes bzw. der Klassifikation. Mögliche Typen sind eigene Schlagworte in deutsch oder englisch, Schlagworte nach der Schlagwortnormdatei (SWD) sowie Klassifikationen nach DNB bzw. RVK. Eine eindeutige Zuordnung ist also durch die Attribute *subject_id*, *metadataset_id* und `type` gegeben, weshalb auch alle drei Attribute gemeinsam einen Schlüsselkandidaten bilden. Weitere Metadatenelemente, die über eine solche Beziehung mit der Relation `pers_metadataset`

[25] Da ein solcher nicht-zusammengesetzter Primärschlüssel verwendet wird, ist es wichtig, durch **unique**`(description_id, metadatset_id)` dieses Tupel zu einem Schlüsselkandidaten (*Alternate Key*) zu machen.

verbunden sind, sind u.a. Daten (mit den Typen Prüfungsdatum, Abgabedatum, Veröffentlichungsdatum) und Titel (mit den Typen Originaltitel, Untertitel, übersetzte Titel). Weitere Informationen finden sich im Anhang im Kapitel C ab Seite 259. Die folgende **Tabelle 10** zeigt noch einmal die Zuordnung der *Dublin Core*-Kategorien zu den gerade vorgestellten Strukturen des Datenbankschemas:

Struktur	DUBLIN-CORE-KATEGORIE
Struktur 1	DC.Identifier
(1:1-Beziehung)	DC.Language
	DC.Type
	DC.Publisher
	DC.Contributor
	DC.Rights
Struktur 2	DC.Description
(direkte N:M-Beziehung)	DC.Sorce
Struktur 3	DC.Date
(indirekte N:M-Beziehung)	DC.Format
	DC.Person
	DC.Relation
	DC.Subjec
	DC.Title

Tabelle 10 *- Zuordnung der DC-Kategorien zu den Strukturen*

Eine zweiter Schwerpunkt der Persistenzrealisierung neben der Spezifikation des Datenbankschemas ist die Entscheidung, auf welche Art von der Anwendung aus auf die Datenbank zugegriffen werden soll, und welche Möglichkeiten der Abarbeitung der DDL/DML-Anfragen dabei genutzt werden soll. Für den Zugriff auf (vor allem relationale) Datenbanken aus Java-Anwendungen heraus existiert das API JDBC[HCF98] [L-JD], welches ein *Call Level Interface*-API ist und starke Ähnlichkeiten zu ODBC aufweist. JDBC erlaubt drei verschiedene Arten, Anfragen an eine Datenbank abzusetzen. Die erste Möglichkeit besteht darin, ein normales `Statement` an die Datenbank abzuschicken. In diesem Fall wird eine normale SQL-Anweisung[26] an die Datenbank geschickt, dort ausgeführt und die Ergebnismenge zurückgesendet. Die Datenbank hat in diesem Fall keine Möglichkeit, die Anfrage zusätzlich zu optimieren. Die zweite Möglichkeit besteht darin, ein sogenanntes `PreparedStatement` zu verwenden. Dieses besteht aus einer SQL-Anweisung, die Platzhalter für später noch hinzuzufügende Parameter enthält[27]. Diese Anfrage wird an die Datenbank gesendet und dort für die Dauer der Anwendung zwischengespeichert und optimiert. In einem zweiten Schritt wird der jeweils aktuelle Parameterwert übermittelt und anschließend die Ergebnismenge zurückgegeben. Dieser Ansatz bietet sich for allem an, wenn sich

[26] wie etwa `select * from pers_metadataset where id = 1`

[27] z. Bsp. `select * from pers_metadataset from id = ?`, mit z. Bsp. `1` für den Platzhalter

in einer SQL-Anfrage immer nur bestimmte Parameter ändern, die zum Beispiel innerhalb einer Schleife eingesetzt werden können. Die dritte Möglichkeit verwendet ein `CallableStatement`, um eine gespeicherte Prozedur innerhalb der Datenbank anzusprechen, wobei bestimmte Ähnlichkeiten zum zweiten Ansatz bestehen. Auch hier werden bestimmte Platzhalter innerhalb der Anfrage durch Parameter ersetzt[28], allerdings ist diese Prozedur auch über die Lebensdauer der Anwendung hinaus innerhalb der Datenbank gespeichert, ist daher immer optimiert und bietet darüberhinaus noch einige weitere Vorteile. Die Persistenzrealisierung des HSSS nutzt in erster Linie gespeicherte Prozeduren, da diese neben der Tatsache, dass sie die performanteste Art darstellen, SQL-Anweisungen auszuführen, auch den Vorteil bieten, mehrere Anweisungen zusammenzufassen. Dies bietet sich insbesondere dann an, wenn eine Anweisung von den Ergebnissen einer vorangegangenen Anweisung abhängig ist. Dieses Problem tritt zum Beispiel bei dem Einstellen der Metadatensätze in die persistente Datenbasis auf. **Listing 9** zeigt beispielhaft eine der gespeicherten Prozeduren, die zu diesem Zweck innerhalb der Datenbank angelegt wurden und von der Anwendung über die JDBC-Schnittstelle angesprochen werden:

```
1  create procedure insert_type @kindofdocument varchar(100), @type_name varchar(100),
2  @type_id numeric(4) output as
3  begin
4      declare @kindofdocument_id numeric(4)
5      select @type_id = (select id from pers_type where name = @type_name)
6      if (@type_id is null)
7      begin
8          select @kindofdocument_id = (select id from pers_kindofdocument where name = @kindofdocument)
9          if (@kindofdocument_id is null)
10         begin
11             insert into pers_kindofdocument (name) values (@kindofdocument)
12             select @kindofdocument_id = (select id from pers_kindofdocument where name = @kindofdocument)
13             if (@kindofdocument_id is null)
14             begin
15                 select @type_id = -1
16             end
17             else
18             begin
19                 insert into pers_type (name, kindofdocument_id) values (@type_name, @kindofdocument_id)
20                 select @type_id = (select id from pers_type where name = @type_name)
21             end
22         end
23         else
24         begin
25             insert into pers_type (name, kindofdocument_id) values (@type_name, @kindofdocument_id)
26             select @type_id = (select id from pers_type where name = @type_name)
27         end
28     end
29 end
```

***Listing 9** : stored procedure*

[28] z. Bsp. `{? = call get_metadataset (?)}`, der zweite Platzhalter würde im Beipiel mit `1` belegt, der erste Platzhalter ist für einen Rückgabeparameter registriert.

Diese gespeicherte Prozedur ist verantwortlich für das Einstellen des Dokumententyps einer Hochschulschrift und erhält dazu von der Anwendung die Parameterwerte für die Dokumentenart und den Dokumententyp. Sie liefert der Anwendung die Identifikationsnummer für den eingestellten Dokumententyp zurück, so dass diese ihn für weitere Datenbankinteraktionen innerhalb des Einstellvorgangs verwenden kann. Zunächst überprüft die gespeicherte Prozedur in Zeile 5, ob der von der Anwendung übermittelte Dokumententyp schon exisitiert. In diesem Fall wäre die Prozedur bereits beendet und würde die entsprechende Identifikationsnummer zurückgeben[29]. Andernfalls ist der Dokumententyp in der Datenbank neu zu erstellen. Da die Relation `pers_type` aber durch einen Fremdschlüssel mit der Relation `pers_kindofdokument` verbunden ist (siehe C im Anhang), benötigt die gespeicherte Prozedur die entsprechende Identifikationsnummer und muss zunächst überprüfen, ob die von der Anwendung übermittelte Dokumentenart bereits in der Datenbank existiert. Wenn das der Fall ist, erstellt die gespeicherte Prozedur ab Zeile 24 ein neues Tupel in der Tabelle `pers_type`, wobei sie für den Fremdschlüssel `kindofdocument_id` den gerade erhaltenen Wert der Anfrage aus Zeile 8 verwendet, und gibt die Identifikationsnummer dieses Tupels zurück. Sollte auch die Dokumentenart noch nicht in der Datenbank existent sein, muss die gespeicherte Prozedur ab Zeile 10 erst den entsprechenden Eintrag in `pers_kindofdocument` erstellen, die Identifikationsnummer der Dokumentenart ermitteln und anschließend die Eintragung in `pers_type` vornehmen[30]. Anschließend wird wieder die Identifikationsnummer des aktuellen Dokumententyps an die Anwendung zurückgegeben.

Nach diesem Muster existieren weitere gespeicherte Prozeduren auch für die anderen *Dublin Core*-Kategorien, die ein relativ übersichtliches und bequemes Einstellen für die Anwendung innerhalb einer Transaktion erlauben. Im Vergleich zu den anderen Möglichkeiten lässt sich dadurch sowohl die Zahl der SQL-Anfragen als auch deren Komplexität deutlich reduzieren. Auch für andere Teile der Anwendung, die mit der Datenbank interagieren und bei denen es auf eine möglichst hohe Performance ankommt, wurden immer gespeicherte Prozeduren verwendet, soweit dies möglich war[31]. Der einzige Nachteil, der durch die Verwendung gespeicherter Prozeduren entsteht, besteht in der fehlenden Standardisierung, so dass sie in den meisten Fällen

[29] Es sind hier zwei Strategien möglich: entweder wird zuerst überpüft, ob der Dokumententyp schon existiert oder es wird sofort probiert, ihn einzustellen. Um den Gesamtablauf zu optimieren, muss diejenige Strategie als erstes gewählt werden, die mit der größten Wahrscheinlichkeit sofort zum Erfolg führt. Im eingeschwungenen Zustand des Systems ist das die Strategie, die davon ausgeht, dass der Dokumententyp bereits existiert.

[30] Dieser aufwendigste Fall kann allerdings nur zweimal auftreten (da im Moment nur zwei Dokumentenarten unterstützt werden).

[31] Leider sind einige Datentypen für große Datenmengen (z. Bsp. `text`, `binary` etc.) nicht in Verbindung mit gespeicherten Prozeduren verarbeitbar.

nicht zwischen verschiedenen Datenbanksystemen austauschbar sind. Im Zuge der zunehmenden Unterstützung des SQL99-Standards durch die meisten modernen Datenbanksysteme ist hier aber Abhilfe in Sicht, insbesondere auch durch die Möglichkeit, gespeicherte Prozeduren in einer plattformunabhängigen Sprache wie Java zu formulieren[32].

[32]Hier sollte man die Möglichkeiten des Sybase Adaptive Server Enterprise 12.5 in diesem Zusammenhang ausloten und die gespeicherten Prozeduren gegebenenfalls in Java neu formulieren.

5.4. Einschätzung

Bei der Teilanwendung *AdminTools* handelt es sich um die umfangreichste und bei weitem komplexeste Teilanwendung innerhalb des Gesamtsystems. Obwohl bereits Ansätze durch die Belegarbeit vorhanden waren, mussten hier im Rahmen der Diplomarbeit erhebliche Investitionen unternommen werden. Die besondere Schwierigkeit bestand dabei darin, die in Abschnitt 3.1 ab Seite 62 spezifizierten Arbeitsabläufe effektiv zu unterstützen, weil sich diese über viele beteiligte interne und externe Institutionen erstrecken. So gab es viele unterschiedlichste Anforderungen an Technologien, Formate und Protokolle, die nur durch die Integration leistungsfähiger externer Programmierschnittstellen, API's und Bibliotheken unterstützt werden konnten. Die Auswahl solcher API's, deren Verständnis sowie die Anpassung und Integration in das bestehende Gesamtsystem erfordern einen hohen Zeitaufwand und ein sauberes Gesamtdesign.

Auch die Teilanwendung *AdminTools* wurde intensiv getestet und hat sich bei der Einstellung vieler Hochschulschriften bereits bewährt. Dabei konnten auch die spezifizierten Arbeitsabläufe vollständig und erfolgreich begleitet werden. Technische Schwierigkeiten gab es zum Teil bei dem Export der Metadaten an externe Institutionen, diese konnten jedoch im weiteren Verlauf relativ schnell behoben werden. Diese (erwarteten) Schwierigkeiten beim Verlassen der Systemgrenzen bestanden vor allem in der Anpassung der Zeichensätze sowie bestimmter Kommunikationsparameter. Zum Teil waren die Exportanforderungen der Institutionen nicht exakt genug, so dass Änderungen im Nachhinein durchgeführt werden mussten, die aufgrund der verwendeten Technologie (XML und XSL) in den meisten Fällen jedoch im Minuten-Bereich lagen und kein Neukompilieren von Anwendungsteilen oder gar ein Neustart des Systems verlangten. Ein weiteres Problemfeld besteht in der automatisierten Textextraktion aus PDF, welches bei bestimmten Dokumenten nicht zufriedenstellend möglich war. Da offizielle Produkte von *Adobe* aber auch keine besseren Ergebnisse zeigten, scheinen diese Probleme genereller Art zu sein und sollten im Vorfeld durch entsprechend sorgfältiges Konvertieren vermieden werden. In diesem Bereich müssen aber auch noch Erfahrungen durch die Praxis bzw. im Austausch mit anderen Universitätsbibliotheken gesammelt werden.

Auch innerhalb dieser Teilanwendung konnten mit Hilfe der beiden Auszubildenden der SLUB zusätzliche Funktionalitäten insbesondere im Administrationsbereich realisiert werden. So ermöglicht insbesondere der Zugriff auf die Konfigurationsstrukturen und die XSL-Stylesheets über die Web-basierte Anwendung flexible Änderungen am Gesamtsystem von überall aus und in der Regel zur Laufzeit. Damit steht den

Mitarbeitern des Referates 5.3 ein mächtiges Werkzeug zur Verfügung. Neben diesen positiven Eigenschaften sollen im Folgenden aber auch Bereiche der Teilanwendung betrachtet werden, die Verbesserungsmöglichkeiten offen lassen bzw. bei denen Erweiterungen sinnvoll sind:

- Die Übersichtseiten, auf denen alle aktuellen bzw. alle eingestellten Metadatensätze präsentiert werden, sollten die Möglichkeit bieten, nur bestimmte Metadatensätze anzuzeigen. Damit würde dem Umstand Rechnung getragen, dass gerade die Zahl der bereits eingestellten Metadatensätze immer stärker anwächst und somit bald eine Menge erreicht wird, die nicht mehr übersichtlich präsentiert werden kann. Dieses Problem könnte durch die Auswahl von Teilmengen (z. Bsp. die 20 „neuesten“ Metadatensätze) bzw. durch Iteration (zeige immer nur 10 Metadatensätze) entschärft werden.
- Die XML-basierten Strukturen sind zwar innerhalb der Teilanwendung sichtbar und können dort auch geändert und zurückgespeichert werden, allerdings erfolgen die Änderungen direkt an den XML-Daten, so dass entsprechende Kenntnisse vorausgesetzt sind. Benutzerfreundlicher wäre eine Indirektion, welche die XML-Struktur vor den Nutzern verbirgt und die entsprechenden Daten stattdessen zum Beispiel Formular-basiert präsentiert. Diese Anwendung sollte die XML-Daten gleichzeitig auch vor „unsinnigen“ Änderungen schützen, die im Extremfall die gesamte Anwendung vorübergehend außer Betrieb setzten könnten.

6. Teilanwendung 3 : HSSS AccessTools / myHSSS

Inhalt des Kapitels:

6.1. Realisierung einer absoluten URL 178

6.2. Objektorientierte Analyse und Design 185

6.2.1. Die Volltextsuchmaschine Lucene 186

6.2.2. Integration von Lucene in die Anwendung 191

6.3. Realisierung 197

6.4. Einschätzung 205

6.1. Realisierung einer absoluten URL

Für die dritte Teilanwendung, die den webbasierten Zugriff auf die Hochschulschriften des HSSS bzw. die Suche und Navigation über deren Metadateninformationen realisieren soll, ist eine funktionierende und dabei möglichst performante und skalierbare Abbildung von absoluten URL's auf reale URL's eine Grundvoraussetzung. Denn jeden Treffer soll diese Teilanwendung als stabile URL präsentieren, erst ein Klick auf eine solche URL führt den Nutzer über die zu realisierende Abbildung zu der (momentan) realen Adresse der entsprechenden Ressource. Eine Abbildung wird also für jeden Zugriff notwendig, was deutlich macht, dass diese Funktion schnell zum Flaschenhals der gesamten Teilanwendung werden kann und deshalb besonders sorgfältig realisiert werden muss.

Eine solche Indirektion ist notwendig, um die langfristige Verfügbarkeit unter einer bestimmten, sich nicht mehr ändernden (absoluten) URL garantieren zu können. Deshalb wird für jede Ressource eine absolute URL erstellt, wobei die Anwendung bzw. das dahinterliegende System dafür garantieren muss, dass diese sich nicht mehr ändert[1]. Selbstverständlich muss die absolute URL pro Ressource bzw. Metadatensatz eindeutig sein. Diese URL wird nun bei jeden Zugriff auf die momentan gerade gültige reale URL abgebildet, typischerweise auf eine wie auch immer konstruierte URL innerhalb des Dokumentenverzeichnisses eines Webservers. Diese URL darf sich im Nachhinein ändern, es muss lediglich die Abbildung von der absoluten auf die reale URL entsprechend angepasst werden, was für die Benutzer transparent bleibt. Sie kennen nur *eine* sich nie mehr verändernde URL, die sie daher auch zitieren oder als Lesezeichen benutzen können.

Um die Abbildung zu realisieren, bedarf es einer Indirektion, die nur von einer entsprechenden Anwendung (im Sinne von Software) realisiert werden kann. Diese Anwendung sollte sich natürlich möglichst gut in das bestehende Gesamtsystem integrieren lassen, so dass sich für die bestehende Architektur eine Java-basierte Anwendung in Form eines Servlets oder eines JSP empfehlen würde. Das Wissen, welches diese Anwendung benötigt, scheint zunächst trivial: eine Menge von Tupeln jeweils bestehend aus der absoluten und der realen URL. Die interessante Frage lautet hier aber: wie liegt dieses Wissen vor? Hier bestehen im Wesentlichen zwei Möglichkeiten, die auf der nächsten Seite aufgeführt sind.

[1] Das gilt zum Beispiel für den Nahmen des entsprechenden Webservers, der als Alias definiert sein sollte und sich auch nicht ändern darf, wenn der physische Rechner gewechselt wird.

1. Das Wissen um die Abbildung liegt explizit vor. Das bedeutet, dass zu jeder absoluten URL an irgendeiner Stelle extern die Information vorliegt, auf welche reale URL diese abzubilden ist. Es existieren zwei persistente URL's: die absolute und die reale.

2. Das Wissen um die Abbildung liegt implizit vor. Das bedeutet, dass keine externe Information zur Abbildung notwendig ist, was voraussetzt, dass alle dafür notwendigen Informationen bereits in der absoluten URL enthalten sind. Es existiert mit der absoluten URL nur eine persistente URL.

Beide Ansätze haben spezifische Vor- und Nachteile. Der erste Ansatz hat den Vorteil, dass es keine einheitliche Abbildungsfunktion gibt[2]. Deshalb sind flexible und auch unterschiedliche Abbildungen bei verschiedenen absoluten URL's möglich. Bei dem zweiten Ansatz muss hingegen eine Berechnungsgrundlage für die Abbildung existieren, die notwendigerweise auch für alle Abbildungen gleich ist. Zwar ließe sich diese Abbildungsfunktion im Nachhinein auch ändern, dann aber nur für *alle* Abbildungen[3]. Der große Vorteil und die besondere Eleganz des ersten Ansatzes besteht hingegen darin, dass er auf keine externen Informationen angewiesen ist und damit auch keine Zeit vergeuden muss, diese zu erlangen. Stattdessen besteht die Abbildung aus einer relativ simplen Berechnung im Hauptspeicher und ist entsprechend performant. Ein noch größere Vorteil besteht darin, dass der Abbildungsvorgang bis auf die Komplexität der Abbildungsfunktion überhaupt nicht von der Menge der möglichen Abbildungen abhängig ist. Die Abbildung konzentriert sich immer auf die momentan abzubildende absolute URL, unabhängig davon, ob diese die einzige oder eine unter 100000 im Gesamtsystem ist. Zusammenfassend lassen sich für die beiden Ansätze also die folgenden hauptsächlichen Vor- und Nachteile ermitteln, die in **Tabelle 11** auf der folgenden Seite dargestellt sind.

[2]da es strenggenommen überhaupt keine formalistische Abbildungsfunktion gibt

[3]Die praktische Relevanz dieser theoretischen Überlegung wird schnell klar, wenn mann sich eine Abbildungsfunktion überlegt, die eine bestimmte Verzeichnisstruktur auf *einem* bestimmten Webserver A berechnet. Es ist nicht ohne weiteres möglich, einen Teil der absoluten URL'S auf URL's des Webservers A abzubilden und gleichzeitig einen anderen Teil auf URL's des Webservers B. Gerade das könnte aber bei einer verteilten Anwendung durchaus Sinn machen.

Ansatz	Vorteile	Nachteile
1. explizite Abbildung (zwei Entitäten)	*+ Flexibilität* Es sind zur selben Zeit mehrere Abbildungsziele denkbar (theoretisch so viele wie Abbildungen). Das bedeutet, dass sich die realen URL's und damit auch der Inhalt (Frontdoors und Dokumente im Präsentationsformat) auf mehrere physische Server verteilen kann. Änderungen der Abbildungen müssen auch nicht für alle URL's gemeinsam ausgeführt werden.	*- Skalierbarkeit* Der Aufwand und damit die Performance der Abbildung ist abhängig von der Menge der möglichen Abbildungen. Damit ist dieser Ansatz zumindest nicht unbegrenzt skalierbar. *- Performance* Die Performance ist generell schlechter als bei dem zweiten Ansatz, da die Infomationen (zu allen Abbildungen!) erst beschafft werden müssen und in irgendeiner Art und Weise (z. Bsp. in einer Hauptspeicherstruktur) vorgehalten werden müssen. Mit zunehmender Größe der Abbildungsmenge nimmt die Performance noch weiter ab, höchstwahrscheinlich progressiv.
2. implizite Abbildung (eine Entität + Abbildungsfkt.)	*+ Skalierbarkeit* Die Abbildung ist abhängig von der Abbildungsfunktion, nicht von der Größe der Abbildungsmenge (vorausgesetzt die Abbildungsfunktion ist so gewählt, dass die Abbildungsmenge nicht den Wertebereich übersteigt). *+ Performance* Die Performance ist direkt und lediglich von der Abbildungsfunktion abhängig und auf keine externen Ressourcen oder Datenstrukturen angewiesen.	*- Flexibilität* Es ist zu einem Zeitpunkt immer nur die Verwendung einer Abbildungsfunktion möglich. Änderungen der Abbildungsfunktionen beziehen sich ebenfalls auf alle Abbildungen. Diese „Alle oder Keiner"-Semantik schränkt die Abbildungsflexibilität stark ein.

Tabelle 11 *- Vor- und Nachteile der Abbildungsansätze*

Obwohl der zweite Ansatz insgesamt praktikabler erscheint, wurde im Rahmen der Diplomarbeit der erste Ansatz verfolgt. Dies liegt insbesondere in den Entwurfsentscheidungen begründet, die zu diesem Zeitpunkt bereits getroffen waren, vor allem im Zusammenhang mit der Bearbeitungsnumer (siehe dazu Abschnitt 3.3.1.1 auf Seite 92) und der Metadatenspezifikation. So wäre die Bearbeitungsnummer bereits sehr gut geeignet, um die Eindeutigkeit der absoluten URL in Bezug auf alle Metadatensätze des Systems zu garantieren. In Hinsicht auf die Unterstützung einer Abbildungsfunktion reicht sie jedoch höchstwarscheinlich nicht aus, da sie außer dem Initialisierungszeitpunkt keinerlei (beabsichtigte) inhärente Informationen trägt. Diese wären aber für

eine praktikable Abbildungsfunktion notwendig[4]. Ohne lesbare, kontextbasierte Teilstrukturen sind die entstehenden realen Verzeichnisstrukturen manuell kaum zu überblicken, was bei einem Ausfall des Systems dazu führen kann, dass die Daten, obwohl vorhanden, schlecht bis gar nicht zu verwalten sind[5]. Wie in Abschnitt 3.3.1.1 beschrieben, wurde daher eine automatisch erstellbare Bearbeitungsnummer bevorzugt, die keine weiteren Informationen benötigt. Gleichzeitig wird vom System eine reale Verzeichnisstruktur (welche die reale URL beeinflusst) konstruiert, die im Notfall eine manuelle Verwaltung erlaubt und damit vom Kontext der entsprechenden Ressource abhängig ist. Ein Beispiel für ein solches Verzeichnis ist

```
<webserver-root>/dissertation/2001/wirtschaftswissenschaften/994670019875-4119/...
```

Diese Struktur besteht aus der Dokumentenart, dem Veröffentlichungsjahr, der veröffentlichenden Fakultät und der Bearbeitungsnummer der Hochschulschrift, wodurch im Extremfall bereits eine rudimentäre Navigation (auch webbasiert) möglich wäre. Durch die absolut eindeutige Bearbeitungsnummer lassen sich diese Verzeichnisse auch ausgezeichnet manuell reorganisieren. Innerhalb der Metadatensatzspezifikation (siehe dazu Abschnitt A.2 im Anhang ab Seite 231) sind deshalb auch zwei Subelemente der Kategorie *DC.Identifier* definiert, was bei Verfolgung des zweiten Ansatzes nicht notwendig wäre.

Aus diesen Gründen bot sich eine direkte Abbildung von zwei persistenten URL's durch das System an, wobei die zu erwartenden Nachteile in den Punkten Performance und Skalierbarkeit möglichst gering gehalten werden sollten. Im Falle des HSSS ist die Anwendung, die die Abbildung realisiert, das Servlet *MappingServlet* im Package `urlmapping`. Dieses Servlet erhält wie alle anderen Servlets des Gesamtsystems durch die Konfigurationsdatei `web.xml` wichtige Konfigurationsprameter. Die **Abbildung 52** auf der folgenden Seite zeigt den groben Ablauf einer URL-Abbildung innerhalb des HSSS.

[4] so wäre es von Vorteil, wenn die Abbildung zu einer kontextbasierten hierarchischen Struktur führen würde, so dass eine relative Streuung innerhalb der einzelnen Hierarchiestufen in Abhängigkeit von bestimmten Eigenschaften der Ressource eintritt. Während eine in etwa gleichverteilte Streuung relativ einfach durch verschiedene modulo-Operationen in Verbindung mit Primzahlen erzeugt werden kann (z. Bsp. `timestamp mod 3` in der ersten Hierarchiestufe, `timestamp mod 5` für die zweite usw.) ist eine kontextbasierte Streuung weitaus schwieriger, weil die Information schon zum Zeitpunkt der Generierung der Bearbeitungsnummer in diese integriert werden müsste.

[5] vorstellbar bei Verzeichnisstrukturen wie
.../2/4/6/22342342izuiziuhn234/...
.../0/2/1/98344564qxcweeiu963/...
usw.
die keine inhaltlichen Bezugspunkte bieten.

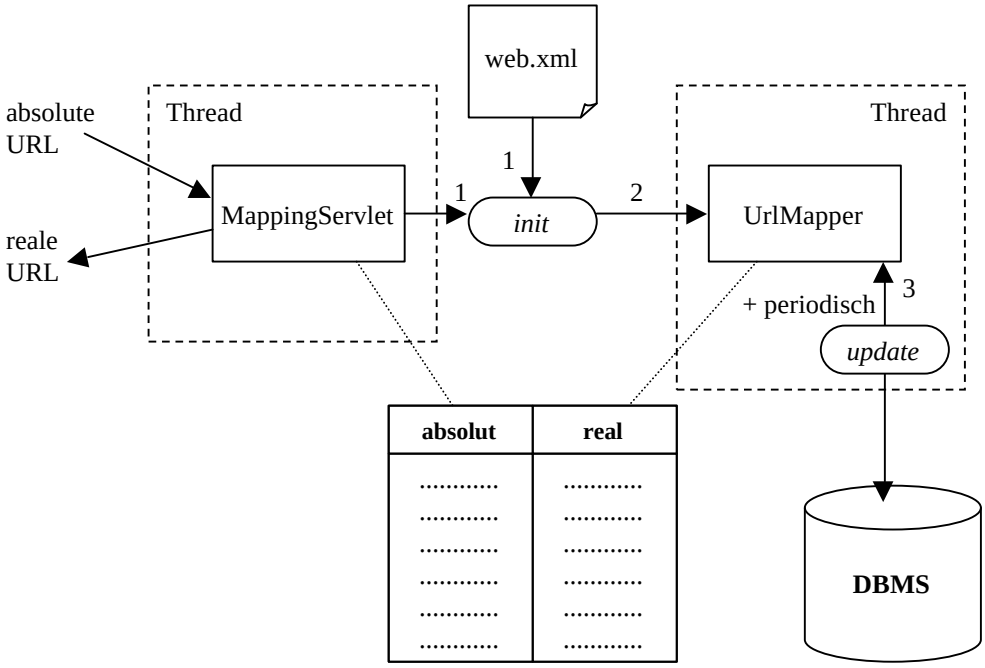

Abbildung 52 *- Vorgang der URL-Abbildung*

Beim ersten Zugriff auf das Servlet `MapingServlet` wird vom Servlet-Container dessen `init()`-Methode aufgerufen. Innerhalb dieser Methode initialisiert das Servlet eine Instanz vom Typ `UrlMapper` innerhalb eines neuen Threads, welche zusätzlich bestimmte Informationen aus der Konfigurationsdatei `web.xml` erhält, die dem `Mappingservlet` zugänglich sind. Außerdem erhält das `UrlMapper`-Objekt eine Referenz auf die (noch leere) Datenstruktur, die das Servlet zur Abbildung von absoluten auf reale URL's verwendet. Sämtliche Zugriffe zum Aufbau und zur Aktualisierung der Datenstruktur, die alle Tupel von absoluten und realen URL's enthält, werden vom `UrlMapper` parallel zum Servlet innerhalb eines eigenen Threads realisiert. Bei der Initialisierung greift dieses Objekt sofort auf eine spezielle gespeicherte Prozedur innerhalb der Datenbank zu, welche die Menge aller Abbildungen zurückliefert sowie das zahlenmäßig größte id-Attribut der Tabelle `pers_metadataset` (siehe Abschnitt 5.3 bzw. Kapitel C). Die Menge der Abbildungen wird in eine interne Datenstruktur (eine Hashtabelle) überführt, die vom selben Typ ist wie die Referenz auf die Datenstruktur des Servlets. Ist der Aufbau der Datenstruktur des `UrlMappers` beendet, wird die durch die Referenz spezifizierte Datenstruktur aktualisiert[6].

[6]Diese Datenstruktur ist durch einen Semaphor synchronisiert und dadurch Thread-sicher. Die Aktualisierung erfolgt durch eine vom Java-API bereitgestellte Methode, welche einer Hashtabelle eine andere Hashtabelle hinzufügt und dabei nur geänderte oder neue Schlüssel berücksichtigt. Im Ergebnis bleibt das alte Objekt und damit auch die Referenz erhalten, so dass diese Aktualisierung beliebig oft wiederholt werden kann.

Der höchste id-Wert wird vom `UrlMapper` benötigt, um den Zugriff auf die persitenten URL's innerhalb der Tabelle `pers_metadatset` optimieren zu können. Diese Optimierung ist deshalb möglich, weil das id-Attribut durch ein automatisches Inkrementieren gebildet wird und Änderungen an Metadatensätzen so realisiert sind, dass der alte Metadatensatz mit Primärschlüssel (id) gelöscht wird und dafür ein neuer Metadatensatz (mit neuem Primärschlüssel) erstellt wird[7]. Dadurch haben Änderungen der Abbildungsmenge (durch neue Metadatensätze oder Änderungen an schon vorhandenen) immer zur Folge, dass sich der Primärschlüsselwert erhöht. Deshalb benötigt der `UrlMapper`, um die Abbildungsmenge zu aktualisieren, nunmehr nur die Tupel, die einem Primärschlüssel zugehörig sind, der größer als der zuletzt zurückgegebene id-Wert ist. Dies führt zu einer wesentlich effizienteren Suche in der Datenbank und einer deutlich kleineren Ergebnismenge und damit zu einer schnelleren Übertragung.

Die Aktualisierungsanfragen unternimmt das `UrlMapper`-Objekt in einem bestimmten Intervall, welches von außen durch einen entsprechenden Konfigurationsparameter in der `web.xml`-Datei beeinflußbar ist. Zusätzlich kann der `UrlMapper` über eine definierte Schnittstelle von außen angesprochen und dazu gebracht werden, eine Aktualisierung sofort durchzuführen. Dies macht insbesondere dann Sinn, wenn eine Anwendung weiß, dass sich die Abbildungsmenge gerade geändert hat[8]. Das Servlet `MappingServlet` hat somit immer auf eine Datenstruktur mit nahezu aktuellen Abbildungen Zugriff und kann die Abbildung durch das Suchen eines eindeutigen Schlüssels (absolute URL) innerhalb der Datenstruktur (Hashtabelle) im Hauptspeicher sehr effizient durchführen. Für die bisherige, allerdings auch nicht sehr große Zahl von persistenten Metadatensätzen funktionierte diese Art der Abbildung sehr stabil und performant. Als Ergebnis wird vom `MappingServlet` der Inhalt der realen URL (also die jeweilige Frontdoor) präsentiert, jedoch immer noch unter der absoluten URL, wie **Abbildung 53** auf der nächsten Seite demonstriert.

[7] also *delete* und *insert* anstelle von *update*

[8] Diese Möglichkeit wird unter anderem von der Teilanwendung *AdminTools* genutzt, wenn ein Metadatensatz gerade erfolgreich eingestellt wurde.

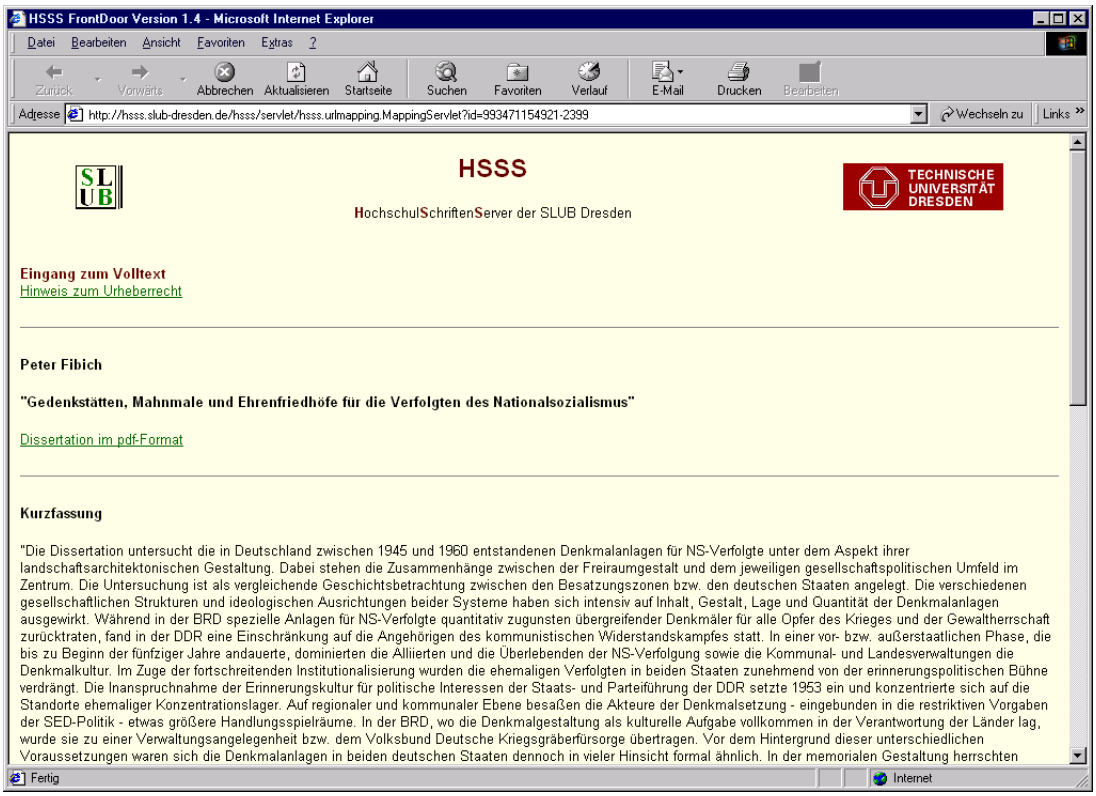
HSSS FrontDoor Version 1.4 - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://hsss.slub-dresden.de/hsss/servlet/hsss.urlmapping.MappingServlet?id=993471154921-2399
Wechseln zu Links
HSSS
HochschulSchriftenServer der SLUB Dresden
TECHNISCHE UNIVERSITÄT DRESDEN
Eingang zum Volltext
Hinweis zum Urheberrecht
Peter Fibich
"Gedenkstätten, Mahnmale und Ehrenfriedhöfe für die Verfolgten des Nationalsozialismus"
Dissertation im pdf-Format
Kurzfassung
"Die Dissertation untersucht die in Deutschland zwischen 1945 und 1960 entstandenen Denkmalanlagen für NS-Verfolgte unter dem Aspekt ihrer landschaftsarchitektonischen Gestaltung. Dabei stehen die Zusammenhänge zwischen der Freiraumgestalt und dem jeweiligen gesellschaftspolitischen Umfeld im Zentrum. Die Untersuchung ist als vergleichende Geschichtsbetrachtung zwischen den Besatzungszonen bzw. den deutschen Staaten angelegt. Die verschiedenen gesellschaftlichen Strukturen und ideologischen Ausrichtungen beider Systeme haben sich intensiv auf Inhalt, Gestalt, Lage und Quantität der Denkmalanlagen ausgewirkt. Während in der BRD spezielle Anlagen für NS-Verfolgte quantitativ zugunsten übergreifender Denkmäler für alle Opfer des Krieges und der Gewaltherrschaft zurücktraten, fand in der DDR eine Einschränkung auf die Angehörigen des kommunistischen Widerstandskampfes statt. In einer vor- bzw. außerstaatlichen Phase, die bis zu Beginn der fünfziger Jahre andauerte, dominierten die Alliierten und die Überlebenden der NS-Verfolgung sowie die Kommunal- und Landesverwaltungen die Denkmalkultur. Im Zuge der fortschreitenden Institutionalisierung wurden die ehemaligen Verfolgten in beiden Staaten zunehmend von der erinnerungspolitischen Bühne verdrängt. Die Inanspruchnahme der Erinnerungskultur für politische Interessen der Staats- und Parteiführung der DDR setzte 1953 ein und konzentrierte sich auf die Standorte ehemaliger Konzentrationslager. Auf regionaler und kommunaler Ebene besaßen die Akteure der Denkmalsetzung - eingebunden in die restriktiven Vorgaben der SED-Politik - etwas größere Handlungsspielräume. In der BRD, wo die Denkmalgestaltung als kulturelle Aufgabe vollkommen in der Verantwortung der Länder lag, wurde sie zu einer Verwaltungsangelegenheit bzw. dem Volksbund Deutsche Kriegsgräberfürsorge übertragen. Vor dem Hintergrund dieser unterschiedlichen Voraussetzungen waren sich die Denkmalanlagen in beiden deutschen Staaten dennoch in vieler Hinsicht formal ähnlich. In der memorialen Gestaltung herrschten
Fertig
Internet

Abbildung 53 *- HSSS Frontdoor*

Auch wenn der gewählte Abbildungsansatz mittelfristig für den HSSS ausreichen wird, sollte er im Rahmen weiterer Ausbaustufen kritisch hinterfragt werden. Zumindest wäre eine Messung des Verhaltens der beiden Ansätze in den Punkten Stabilität und Performance bei verschiedenen, auch sehr großen Abbildungsmengen sinnvoll, um zu zeigen, wie groß die vermuteten Unterschiede zugunsten des zweiten Ansatzes tatsächlich ausfallen. Allerdings dürfte eine Umstellung des Systems nicht ganz trivial sein, da sie höchstwahrscheinlich in einen Bereich vorstößt[9], der nicht umgestellt werden darf: bereits veröffentlichte absolute URL's. Deshalb müsste eine zukünftige Abbildung in diesem Fall beide „Versionen" der absoluten URL an einem ausgezeichneten Merkmal erkennen und unterschiedlich mit dem jeweils passenden Abbildungsansatz abbilden.

[9]dann nämlich, wenn sich die Struktur der absoluten URL ändern muss, z. Bsp. infolge einer Änderung der Bearbeitungsnummern

6.2. Objektorientierte Analyse und Design

Die dritte Teilanwendung soll die Suche und die Navigation innerhalb der Metadatensätze und möglichst auch innerhalb des Volltextes der Hochschulschriften des HSSS ermöglichen sowie die Darstellung der Ergebnisse realiseren. Die Suche und die Navigation sind dabei auf drei verschiedene Arten realisierbar:

- **Direkt innerhalb der Datenbank ohne zusätzliche Hilfsmittel**
  Dieser Ansatz ist der am schwierigsten zu realisierende, zumindest wenn die Suche einigermaßen effizient und mit einer brauchbaren Performance erfolgen soll. Insbesondere verlangt dieser Ansatz ein auf die Suche abgestimmtes Schema. Die Volltextsuche ist nur sehr eingeschränkt realisierbar.

- **Innerhalb der Datenbank unterstützt durch zusätzliche Datenbankfunktionalitäten**
  Vielfach enthalten moderne Datenbanken zusätzliche Funktionalitäten in Form von integrierten Volltextsuchmaschinen, oder diese Funktionalitäten sind nachrüstbar. Sie erlauben in der Regel eine optimale Integration des vorhandenen Datenbankschemas. Selbstverständlich haben solche Datenbanken bzw. die entsprechenden Volltextmodule ihren Preis.

- **Außerhalb der Datenbank durch die Nutzung einer externen Volltextsuchmaschine**
  Externe Volltextmaschinen sind auf die Suche innerhalb von indexierten Texten spezialisiert und sollten deshalb exakte Suchergebnisse mit einer guten Performance liefern. Leider besteht hier das Problem, dass man diese Komponente zusätzlich in das System integrieren und die Synchronität von Datenbasis und Volltextindex gewährleisten muss.

Der erste Ansatz verlangt selbst ohne Realisierung der Suche in den Volltexten einige Investitionen in das Datenbankschema, welches dadurch zusätzlich verkompliziert würde. Um beispielsweise auch bei großen Datenmengen in einer annehmbaren Geschwindigkeit die Beschreibungen oder Titel der Hochschulschriften nach einzelnen Worten durchsuchen zu können, müsste auch eine separate Tabelle für einzelne Worte geschaffen werden[10]. Zusätzlich müsste es eine Tabelle mit Stopworten geben[11].

[10] Die einzelnen Worte sind Primärschlüssel und werden indiziert. Andere Ansätze, wie die Nutzung des SQL-Operators *like* sind aufgrund mangelnder Performance nicht praktikabel.

[11] Zu beachten ist hier, dass eine korrekte Normalisierung verlangt, dass sich beispielsweise eine Beschreibung nur aus Fremdschlüsseln der Wort-Tabelle und der Stopwort-Tabelle zusammensetzt.

Aufgrund seines Aufwands sollte dieser Ansatz nur im Notfall verfolgt werden. Der zweite Ansatz ist für den HSSS leider nicht umsetzbar, da die zusätzlich lizenzierbare integrierte Volltextsuchmaschine in der Landeslizenz für Sybase nicht enthalten ist. Weitere Ausbaustufen, die über den Status eines Prototypen hinausgehen, sollten diese Möglichkeit in Betracht ziehen und die Fähigkeiten und möglichen Vorteile einer integrierten Volltextsuche genau eruieren. Die gegenwärtige Version des HSSS verwendet stattdessen eine externe Volltextsuchmaschine. Hier konnte mit dem Java-basierten Open-Source-Produkt *„Lucene“* (siehe auch Abschnitt D) eine Komponente gefunden werden, die sich relativ nahtlos in das bestehende Gesamtsystem integrieren ließ und einige sehr positive Eigenschaften bietet. Der folgende Abschnitt beschäftigt sich etwas näher mit dieser Volltextsuchmaschine.

6.2.1. Die Volltextsuchmaschine Lucene

Bei *„Lucene“* handelt es sich um eine in Java geschriebene Open Source - Volltextsuchmaschine, die durch ihr sauberes Design und die gute Integrationsfähigkeit besticht. Sie ermöglicht die wichtigsten Suchmöglichkeiten wie etwa boolische Suche, exakte Suche („phrase queries“) und Feldsuchen und hebt sich dabei durch einen skalierbaren, sehr performanten Indexaufbau[12] vor, welcher durch spezielle Strategien inkrementell genauso schnell ist wie im Batch-Modus. Dabei stellt *Lucene* nur minimale Anforderungen an Speicher und Hauptspeicher[13] [L-LUb]. *Lucene* wird von einer Reihe von Entwicklern gepflegt und weiterentwickelt und höchstwahrscheinlich noch in diesem Jahr von der ASF (*Apache Software Foundation*) - eine der größten Open Source-Initiativen - unterstützt, die damit ihr Portfolio im Bereich Web-Services um ein interessantes Produkt bereichert[14]. Das API von *Lucene* verteilt sich auf mehrere Packages, die jeweils getrennte Funktionsbereiche lose gekoppelt realisieren:

- `com.lucene.util`
  Dieses Package enthält einige Hilfsklassen und Datenstrukturen, etwa `BitVector` oder `PriorityQueue`

Will mann allerdings auf die gesamte Beschreibung zugreifen, muss diese durch Verbünde erst wieder relativ aufwendig zusammengesetzt werden (d.h. es existiert ein Zielkonflikt zwischen verschiedenen Zugriffsarten).

[12] laut eigenem Datenblatt etwa 200MB/Stunde auf einem Pentium II/266MHz

[13] etwa 1MB Heap

[14] Dieser Schritt ist für die Zukunft von *Lucene* sehr wichtig, da somit gegenüber potentiellen Interessenten die Weiterentwicklung und die finanzielle Unterstützung in einer ganz anderen Art und Weise glaubhaft versichert werden kann.

- `com.lucene.store`
  Dieses Package enthält alle für das Speichern von Indizes und Segmenten notwendigen Klassen, darunter auch viele abstrakte Klassen, die entsprechend erweitert werden können, wie etwa `InputStream`, `OutputStream` oder `Directory`.

- `com.lucene.document`
  Dieses Package enthält alle Klassen, die das Dokumentenmodell von Lucene realisieren, das sind vor allem `Document` und `Field`.

- `com.lucene.analyses`
  Dieses Package definiert eine abstrakte Schnittstelle vom Typ `Analyzer`, welche Texte von einer Instanz des Typs `java.io.Reader` in eine Instanz vom Typ `TokenStream` umwandelt (eine Menge geordneter `Token`). Außerdem werden eine Reihe einfacher Spezialisierungen bereitgestellt, wie etwa `StopAnalyzer` und eine Reihe von Filtern, wie etwa `PorterStemFilter`.

- `com.lucene.index`
  Dieses Package realisiert den Zugriff auf einen Index. Dafür stehen die Klassen `IndexWriter` und `IndexReader` zur Verfügung.

- `com.lucene.query`
  Dieses Package stellt Datenstrukturen zur Verfügung, um bestimmte Anfragearten abbilden zu können (z. Bsp. `TermQuery`, `PhraseQuery` oder `BooleanQuery`). Außerdem enthält es die abstrakte Klasse `Searcher` als Ausgangsbasis für alle Klassen, die Suchanfragen (`Query`) in eine Sammlung von Treffern (`Hits`) umwandeln können.

- `com.lucene.queryParser`
  Dieses Package enthält die Implementation eines `QueryParser` für die Anfragen an *Lucene*.

Lucene benutzt ein recht simples, aber flexibles Dokumentenmodell, um Indexinformationen zu repräsentieren. Ein Dokument (`Document`) ist dabei eine Sammlung von zusammengehörigen Informationen, die in verschiedenen Feldern (`Field`) innerhalb des Dokuments abgelegt werden können. Ein Feld ist ein Schlüssel-Wert-Paar, bei dem der Wert aus einem `String` oder einem `Reader` bestehen kann[15]. Außerdem sind jedem Feld drei boolische Werte zugeordnet, die dessen Nutzung beim Indizieren spezifizieren. Der boolische Wert `isIndexed` signalisiert, dass der Inhalt dieses Feldes bei Anfragen durchsucht wird, `isStored` bedeutet, dass der Inhalt des Feldes innerhalb des Indexes gespeichert wird, so dass eine vollständige Kopie des Feldes verfügbar

[15] Da ein Dokument aus einer Reihe solcher Schlüssel-Wert-Paare (Felder) besteht, ist es genaugenommen eine besondere Hashtabelle.

ist, wenn das Dokument als Treffer zurückgegeben wird, und `isTokenized` gibt an, ob der Inhalt des Feldes in einzelne Token aufgebrochen und jeweils einzeln indiziert werden soll, oder ob der Inhalt als ganzes indiziert und nicht in Token unterteilt wird[16]. Außerdem stellt *Lucene* ein spezielles Feld für den Fall zur Verfügung, dass ein Datum gespeichert und später durchsucht werden soll. Mit diesem Dokumentenmodell lassen sich Daten von beliebigen Quellen zu einem neuen Datenkonstrukt zusammenfassen, das speziell der Volltextsuche dienen soll.

Die entstehenden Dokumente werden von *Lucene* in einer bestimmten, optimierten Form in binäre Strukturen (Indizes) umgewandelt. Die API ermöglicht dabei, die Indizes auf verschiedene Weise persistent oder transient anzulegen. Normalerweise wird für jeden Index ein eigenes Verzeichnis verwendet, dass verschiedene Dateien enthält, die jeweils ein Segment des Indexes repräsentieren. Ein Dokumentenindex besteht bei *Lucene* - vereinfacht gesehen - aus einer Menge von *Termen*, die wiederum durch ein Tupel aus Feld und Token[17] definiert sind [L-LUc]. Die Hauptaufgaben bestehen dabei darin, die Dokumentenindizes zu erstellen und diese gegebenenfalls zu mischen. Das Besondere an *Lucene*'s Index-Strategie ist dabei, dass das inkrementelle Indizieren von Dokumenten genauso performant unterstützt wird wie das vollständige Indizieren aller Dokumente. Dabei benutzt *Lucene* einen speziellen, einen Stapel von Segmenten verwaltetenden Algorithmus, um einzelne Indexsegmente flexibel in größere Segmente zu mischen[L-LUd]:

1. erstelle einen Index für jedes neu dazukommende Dokument

2. lege den neuen Index auf den Stapel

3. führe den eigentlichen Mischungs-Algorithmus aus:

```
Es sei M der Mischungsfaktor und K = unendlich sowie size eine lokale Variable
for(size=1;size<K;size=size*M){
    if(es sind M Indizes mit size Dokumenten oben auf dem Stapel){
        nimm diese Indizes vom Stapel;
        mische sie in einen einzigen Index;
        lege diesen Index zurück auf den Stapel;
    } else {
        abbrechen;
    }
}
```

[16] sinnvoll zum Beispiel bei URL's etc.

[17] Teil des Inhalts oder der gesamte Inhalt eines indizierten Feldes, abhängig von `isTokenized`

Der Mischungsfaktor ist als entscheidende Größe den jeweiligen Verhältnissen anzupassen. Zur Optimierung werden die nur aus einem Dokument bestehenden Indizes im Hauptspeicher gehalten. Durch diese Mischungsstrategie ist inkrementelles Indexieren performant möglich, trotzdem wird die Segmentzahl bei gut gewähltem Mischungsfaktor relativ gering gehalten. **Abbildung 54** verdeutlicht den Algorithmus noch einmal beispielhaft:

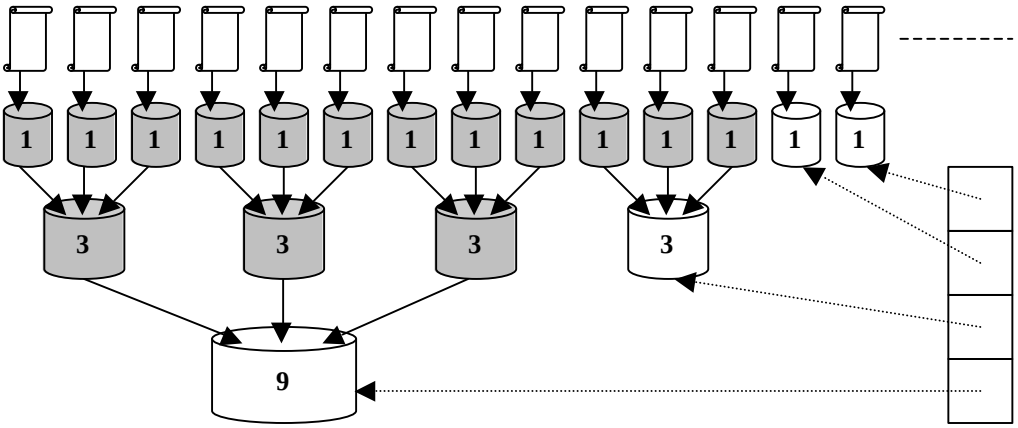

Abbildung 54 *- Lucene Index-Merge*

In diesem Fall wurde der Mischungsfaktor `M`=3 gesetzt, und es haben schon 5 Mischungen stattgefunden. Die grau schattierten Indizes sind auf diese Weise bereits gelöscht (zusammengefasst) worden. Interessant ist dabei, dass eine Kette von Zusammenfassungen entstehen kann. So befanden sich nach dem Ablegen des neunten Dokumentenindexes `M`=3 Einzeldokumentenindizes auf dem Stapel sowie zwei Indizes mit drei Dokumenten. Die drei Einzeldokumentenindizes wurden zunächst zu einem Index mit drei Dokumenten zusammengefasst und dieser wieder auf den Stapel geschoben. Dadurch entstand die Situation, dass sich `M`=3 Indizes mit jeweils drei Dokumenten auf dem Stapel befanden, so dass diese ebenfalls zusammengefasst und als Index für 9 Dokumente auf den Stapel zurückgeschoben wurden.

Ist der Vorgang der Indexierung vorerst abgeschlossen, kann von *Lucene* eine weitere Optimierung vorgenommen werden, die durch die Operation `optimize()` der Klasse `IndexWriter` eingeleitet wird und eine schnellere Suche ermöglicht. Allerdings ist diese Optimierung eine teure Operation, die nur dann aufgerufen werden sollte, wenn der Index eine Zeit lang unverändert bleibt und nur für die Suche verwendet wird. Die Suchanfragen, die jetzt an *Lucene* für den erstellten Index übermittelt werden können, müssen einer bestimmten Syntax genügen. Die Syntaxdefinition für Suchanfragen wird auf der folgenden Seite gezeigt.

Query	::=	**Clause** ([**Conjunction**] **Clause**)*
Conjunction	::=	'AND' | 'OR' | '||'
Clause	::=	[**Modifier**] [**FieldName**':'] **BasisClause** [**Boost**]
Modifier	::=	'-' | '+'| '!'| 'NOT'
BasisClause	::=	(**Term** | **Phrase**) | **PrefixQuery** | '('**Query**')'
PrefixQuery	::=	**Term**'*'
Phrase	::=	'"'(**Term**)+'"'
Boost	::=	'^ '(<Dezimalzahl>)+'.'(<Dezimalzahl>)+
Term	::=	<gesuchtes Wort bzw. Token>
FieldName	::=	<Name eines indizierten Feldes>

Eine Suchanfrage, die dieser Syntax entspricht, wird von einem `QueryParser` in eine interne Repräsentation umgewandelt, die dann an eine Instanz der Klasse `IndexSearcher` übergeben werden kann. Diese nimmt die eigentliche Suche im Index vor und liefert eine Instanz vom Typ `Hits`, welche die Ergebnismenge repräsentiert. Mit Hilfe dieser Ergebnismenge können die einzelnen Treffer ausgewertet und auf entsprechende Feldinhalte zugegriffen werden. Die folgende Aufzählung fasst noch einmal grob zusammen, welches die wichtigsten Schritte sind, um mit *Lucene* zu interagieren[18]:

1. Erstellen eines Index
   a) Erstellen von Objekten des Typs `Document`
   b) Hinzufügen von Feldern (`Field`)
   c) Erstellen eines `IndexWriter-Objekts und Hinzufügen der Dokumente durch addDocument()`
2. Erstellen der Suchanfrage und Übermittlung an Lucene
   a) Aufrufen von `QueryParser.parse()`, liefert ein `Query`-Objekt
   b) Erstellen einer `IndexSearcher`-Instanz
   c) Aufrufen der `search()`-Methode des `IndexSearcher`, liefert ein Ergebnis vom Typ `Hits`
3. Auswerten des Ergebnisses
   a) Zugriff auf die einzelnen Treffer mit `score(int i)` (Rang des i-ten Treffers) bzw. `doc(int i)` (Dokumenteninformationen des i-ten Treffers)

[18] dabei wird der Fall angenommen, dass auch noch ein Index neu zu erstellen ist

6.2.2. Integration von Lucene in die Anwendung

Die Volltextsuchmaschine *Lucene* konnte aufgrund ihres sauberen objektorientierten Designs sehr gut in die gesamte Anwendung integriert werden. Innerhalb des Packages `hsss.utilities` enthält das Package `lucene` alle Klassen, die den Zugriff auf die Volltextsuchmaschine kapseln. **Abbildung 55** zeigt das Klassendiagramm dieses Packages:

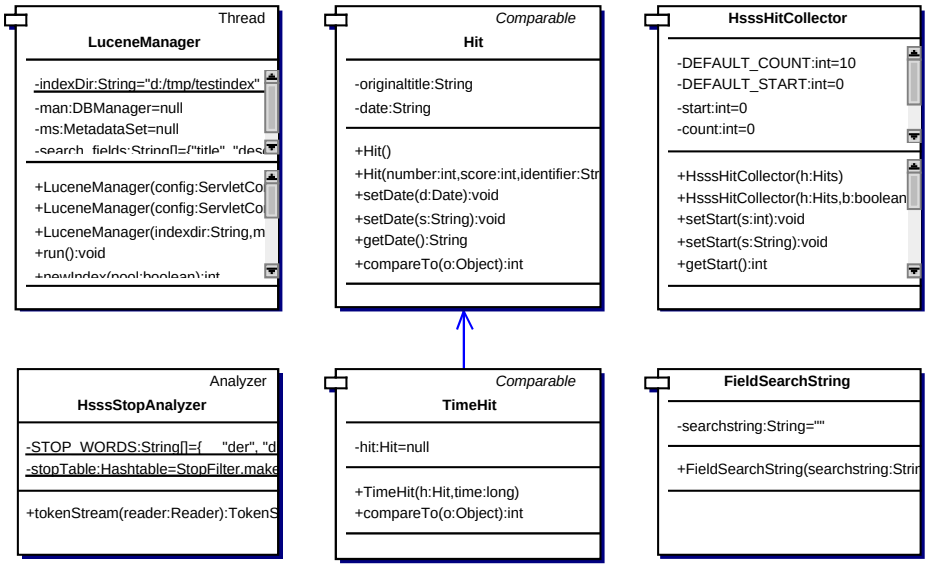

Abbildung 55 *- HSSS Package "lucene"*

Zentraler Zugriffspunkt auf die Volltextsuchmaschine für die Anwendung ist die Klasse `LuceneManager`, die auch innerhalb eines separaten Threads ablaufen kann[19]. Diese Klasse stellt Methoden für alle notwendigen Interaktionen bereit, zum Beispiel `newIndex()`, `addDocToIndex()`, `search()` oder `searchFields()` (u.a.). Von den Suchmethoden wie `search()` wird ein `HsssHitCollector`-Object zurückgegeben, das die Suchergebnisse enthält und einige weitergehende Funktionalitäten unterstützt, die im Anschluss erläutert werden. Ein einzelner Treffer wird durch die Klasse `Hit` gekapselt. `TimeHit` enthält die Klasse `Hit` und verändert deren Verhalten und definiert insbesondere eine andere Wertigkeit der einzelnen Treffer zueinander, so dass eine neue Reihenfolge bei Treffersammlungen entsteht. Die Klasse `FieldSearchString` kapselt eine Teil-Suchanfrage bei der Feldsuche. Sie besteht aus der eigentlichen Suchanfrage für das jeweilige Feld sowie zwei boolischen Werten, die das Auftreten dieses Suchausdrucks im Verhältnis zu dem anderer Felder beschreiben.

[19] sinnvoll beispielsweise, wenn der gesamte Index vollständig neu erstellt werden soll

Dabei handelt es sich um die Parameter:

- `required`
  bestimmt, ob der Teilausdruck im Gesamtergebnis vorkommen muss

- `prohibited`
  bestimmt, ob der Teilausdruck im Gesamtergebnis nicht vorkommen darf

Damit ergeben sich die drei Kombinationen **muss** (`required=true`, `prohibited=false`), **kann** (`required=false`, `prohibited=false`) und **nicht** (`required=false`, `prohibited=true`)[20]. `HsssStopAnalyzer` ist eine Klasse, welche die abstrakte Klasse `Analyzer` aus dem Package `com.lucene.analyses` erweitert (siehe Abschnitt 6.2.1) und um eine deutsche Stopwortliste ergänzt. Eine Klasse, die einen deutschen Stemming-Algorithmus implementiert, konnte aufgrund der knappen Zeit im Rahmen der Diplomarbeit nicht mehr realisiert werden und verbleibt daher als Erweiterungsmöglichkeit für spätere Ausbaustufen des HSSS.

Der `LuceneManager` wird als Erstes dafür genutzt, um einen Index zu erstellen. Voraussetzung dafür ist die Definition eines Dokuments mit einer bestimmten Feldstruktur. **Tabelle 12** zeigt die für die dritte Teilanwendung definierten Dokumentenfelder und deren Inhalt:

Feld	Inhalt
identifier	die absolute URL der Hochschulschrift
fulltext	der Volltext der Hochschulschrift
title	Originaltitel, alternative Titel und übersetzte Titel der Hochschulschrift
subject	Schlagworte (eigene, SWD) und Klassifikationen (DNB, RVK) der Hochschulschrift
description	Beschreibungen der Hochschulschrift
creator	der (die) Autor(en) der Hochschulschrift
advisor	der Betreuer der Hochschulschrift
referee	die Gutachter der Hochschulschrift
date	das Veröffentlichungsdatum der Hochschulschrift
type	der Dokumententyp der Hochschulschrift

***Tabelle 12** - Dokumentenfelder und Inhalte*

[20] Da die vierte Kombination `required=true` und `prohibited=true` nicht sinnvoll und daher verboten ist

Bei Feldern, die mehrere logisch zu trennende Informationen beinhalten (wie etwa *title* oder *subject*) oder die eine Menge von Werten enthalten können (beispielsweise *subject* oder *referee*) werden die einzelnen Entitäten durch eine Separator-Zeichenfolge getrennt, um sie später auf einfache Weise logisch trennen zu können[21]. Für Dokumente mit den beschriebenen Feldern wird vom `LuceneManager` ein Index erstellt, der als eigenes Verzeichnis realisiert ist (durch `FSDirectory`). Es existieren zwei Möglichkeiten, den Volltextindex des HSSS zu beeinflussen: zum einen kann der Index aus allen in der Datenbasis enthaltenen Dokumenten komplett neu aufgebaut werden, zum anderen können an einem vorhandenen Index inkrementelle Veränderungen (Dokument ändern, neues Dokument hinzufügen) vorgenommen werden. Beide Aktionen sind unabhängig davon, ob die Datenbasis dateibasiert oder datenbankbasiert realisiert ist. Im ersten Fall würde man den *LuceneManager* innerhalb eines eigenen Threads instanzieren, um die Gesamtanwendung möglichst wenig zu belasten. Dieser Komplettaufbau sollte in regelmäßigen Abständen im Rahmen der Wartung des HSSS durchgeführt werden[22]. Das inkrementelle Hinzufügen von Dokumenten zum Index kann über eine öffentliche Methode ebenfalls von außen geschehen und wird vor allem von der Teilanwendung *AdminTools* im Zuge des Einstellens von Metadatensätzen genutzt. Da die Häufigkeit und der Zeitpunkt der Aufrufe dieser Methoden für den `LuceneManager` nicht vorauszusehen sind, wird der Index nach jeder Operation optimiert.

Der zweite Funktionsbereich des `LuceneManagers` ist die Realisierung der von den *AccessTools* zu unterstützenden Suchen. Dazu bietet diese Klasse eine Reihe von Methoden an, welche die internen Funktionen von *Lucene* transparent für die Anwendung kapseln. Zu realisieren sind dabei vor allem die Volltextsuche sowie die Feldsuche. Um die Ergebnisse vom `LuceneManager` zu erhalten, muss die dritte Teilanwendung lediglich die passende Methode aufrufen und die entsprechenden Suchanfragen übergeben, die sie als HTTP-Parameter erhält. Auch die dritte Teilanwendung (und damit die Suche) ist Session-basiert realisiert, was nicht notwendig ist, aber eine Reihe von Vorteilen bietet, die im weiteren Verlauf des Abschnitts deutlich werden sollten. Da der Nutzer bei großen Ergebnismengen nicht immer gleich alle Treffer sehen möchte, erlaubt die Implementation der *AccessTools*, lediglich Teilmengen des Gesamtergebnisses zurückzugeben (sogenanntes *Paging*). Dies wird insbesondere durch die Klasse `HsssHitCollector` realisiert, welche die Treffermengen verwaltet. Die Anzahl der Treffer, die auf einmal zurückgegeben werden sollen und andere Parameter sind innerhalb der Benutzeroberfläche der *AccessTools* (*myHSSS*) einstellbar, sie wird näher im nachfolgenden Abschnitt 6.3 vorgestellt.

[21] Dies wird bei der Präsentation von Ergebnissen notwendig. Die Seperator-Zeichenfolge wurde mit „xxx“ gewählt und wird ebenfalls indexiert (steht nicht in der Stopwortliste) und wird auf jeden Fall angehängt (auch bei einer Menge von eins). Das hat den Vorteil, dass auf diese Weise recht einfach nach allen Dokumenten innerhalb des Index gesucht werden kann.

[22] möglichst zu einer Zeit, wo keine oder nur sehr wenige Nutzer die Suchfunktionen von Lucene über diesen Index nutzen

Jede der vom `LuceneManager` angebotenen Suchmethoden liefert ein Objekt vom Typ `HsssHitCollector` zurück. Dieses Objekt wird vom `RequestResponseManager` der *AccessTools* der Session als Attribut hinzugefügt und ist somit auch innerhalb der *View* durch die entsprechenden JSP's verfügbar. Dabei wird eine Strategie verfolgt, die eine besonders ressourcenschonende und performante Realisierung der Suche sowie der Teilmengenbildung erlaubt. Dazu wird am Ende der Suche der Konstruktor der Klasse `HsssHitCollector` mit dem Suchergebnis-Objekt vom Typ `Hits` und der aktuellen Teilmengengröße aufgerufen. Der `HsssHitCollector` legt das `Hits`-Objekt intern als Attribut ab. Zusätzlich existieren Attribute, die das Navigieren innerhalb der einzelnen Teilmengen erlauben [23], die den Detaillierungsgrad der Ergebnisrepräsentation bestimmen und die als Datencontainer für bereits gelieferte Teilergebnisse fungieren[24].

Die *View* ruft nun die zentrale Methode `getPageSet()` auf, welche die erste Teilmenge liefert (oder auch die gesamte Ergebnismenge). Diese Methode ist im **Listing 10** auf der folgenden Seite abgebildet. Mit dem innerhalb des `HsssHitCollector` enthaltenen `Hits`-Objekt stehen lediglich die Rangnummer, die Wertigkeit (`score`) und eine Referenz auf das jeweilige Dokument für die einzelnen Treffer der Ergebnismenge zur Verfügung. Insbesondere bei großen Treffermengen wäre es ineffizient, sofort alle sich in den Dokument-Referenzen verbergenden Informationen in entsprechende Datenstrukturen abzulegen. Ein solches Vorgehen ist aus dem Grund problematisch, weil das Auffinden der gespeicherten Feldinformationen pro Dokument im Index eine teure Operation ist, welche die Antwortzeit auf die Suchanfrage signifikant erhöht. Außerdem ist die entstehende Datenstruktur, welche diese Feldinformationen der Dokumente enthält, um ein Vielfaches größer als das originale `Hits`-Objekt. Will der Nutzer nun auch noch nur einen Bruchteil des Gesamtergebnisses wirklich sehen, sind die Aufwände für alle ignorierten Dokumente umsonst unternommen worden. Deshalb bietet sich ein differenziertes Vorgehen an, wie es durch die Methode `getPageSet()` realisiert wird.

Der Klasse `HsssHitCollector` sind die augenblicklich aktuellen Werte für `start` (Zeiger auf den ersten Treffer der Teilmenge) und `count` (Größe der Teilmenge) bekannt. Als erstes wird durch den Aufruf der Methode `isPageCached` herausgefunden, ob es sich bei der aktuell geforderten Teilmenge um eine Teilmenge handelt, die schon einmal geliefert wurde und die sich in diesem Fall in der Datenstruktur `results` (Cache) befinden würde. Ist dies der Fall, brauchen nur noch die entsprechenden Objekte des Typs `Hit` in die zurückzuliefernde Ergebnismenge kopiert zu werden.

[23] `start` - zeigt auf den jeweils ersten Treffer der Teilmenge, `count` - gibt die aktuell gewählte Größe der Teilmenge an

[24] `results` - fungiert quasi als Cache für schon gezeigte Teilmengen, deren Dokumenteninformationen vollständig vorhanden sind (`SortedMap`)

```
1  ...
2  public SortedSet getPageSet()
3  {
4      SortedSet result = new TreeSet();
5      if (isPageCached())
6      {
7          int ende = Math.min((getStart()+getCount()), getLength());
8          for (int i=getStart(); i<ende; i++)
9          {
10             try
11             {
12                 Hit h = (Hit)results.get(new Integer(i));
13                 result.add(h);
14             } catch (Exception ex)
15             {
16                 Debug.logError(this, ex);
17             }
18         }
19     } else
20     {
21         int ende = Math.min((getStart()+getCount()), getLength());
22         for (int i=getStart(); i<ende; i++)
23         {
24             try
25             {
26                 int score = new Float(hits.score(i)*100).intValue();
27                 int number = i+1;
28                 Hit hit = new Hit(number, score, hits.doc(i).get("identifier"));
29                 hit.setAuthor(getAuthor(hits.doc(i).get("creator")));
30                 hit.setDate(DateField.stringToDate(hits.doc(i).get("date")));
31                 hit.setOriginalTitle(getOriginalTitle(hits.doc(i).get("title")));
32                 hit.setType(hits.doc(i).get("type"));
33                 if (detailed)
34                 {
35                     hit.setTitle(getTitle(hits.doc(i).get("title")));
36                     hit.setDescription(getDescription(hits.doc(i).get("description")));
37                 }
38                 result.add(hit);
39                 results.put(new Integer(i), hit);
40             } catch (Exception ex)
41             {
42                 Debug.logError(this, ex);
43             }
44         }
45     }
46     return result;
47 }
48 ...
```

***Listing 10** : Realisierung des „Pagings“*

Andernfalls müssen für die entsprechenden Treffer neue `Hit`-Objekte erstellt werden (ab Zeile 28). Die dafür notwendigen Informationen werden aus dem `Hits`-Objekt gewonnen. Die Feldinhalte werden zum Teil in Abhängigkeit des gewählten Detaillierungsgrades gesetzt und teilweise noch entsprechend behandelt[25]. Anschließend wird

[25]so liefert `getDescription()` nur die ersten 1000 Zeichen einer Beschreibung, zusätzlich werden die Separator-Zeichen („xxx“) entfernt und adäquat ersetzt (z. Bsp. Zeilenumbruch etc.).

der entstandene `Hit` sowohl der zurückzuliefernden Ergebnismenge (`result`) als auch der Cache-Datenstruktur (`results`) hinzugefügt (Zeile 38 und 39). Ist dieser Vorgang für alle Dokumente der aktuellen Teilmenge abgeschlossen, wird die Ergebnismenge (`result`) zurückgegeben und kann von der *View* entsprechend präsentiert werden. Sollte diese Teilmenge noch einmal angefordert werden, kann sie direkt aus dem Cache (`results`) geholt werden und damit noch schneller zurückgeliefert werden. Dadurch erreicht der hier vorgestellte Ansatz eine sehr gute Skalierung in Abhängigkeit der vom Autor wirklich geforderten Teilergebnisse und eine sehr gute Performance.

6.3. Realisierung

Nachdem im vorangegangenen Abschnitt mit der Vorstellung der Volltextsuchmaschine *Lucene* und mit deren Integration in die Gesamtanwendung serverseitige Aspekte der dritten Teilanwendung behandelt wurden, soll sich dieser Abschnitt speziell mit der Clientseite beschäftigen. Dabei stehen die Realisierung der Benutzerschnittstelle und insbesondere die Umsetzung der in Abschnitt 3.2.1 aufgestellten Architekturvisionen drei und vier im Vordergrund. Für die Prototyp-Version der Benutzerschnittstelle der dritten Teilanwendung wurde der Name *„myHSSS“* gewählt. Anhand von **Abbildung 56** soll zunächst der allgemeine Aufbau der Benutzeroberfläche erläutert werden:

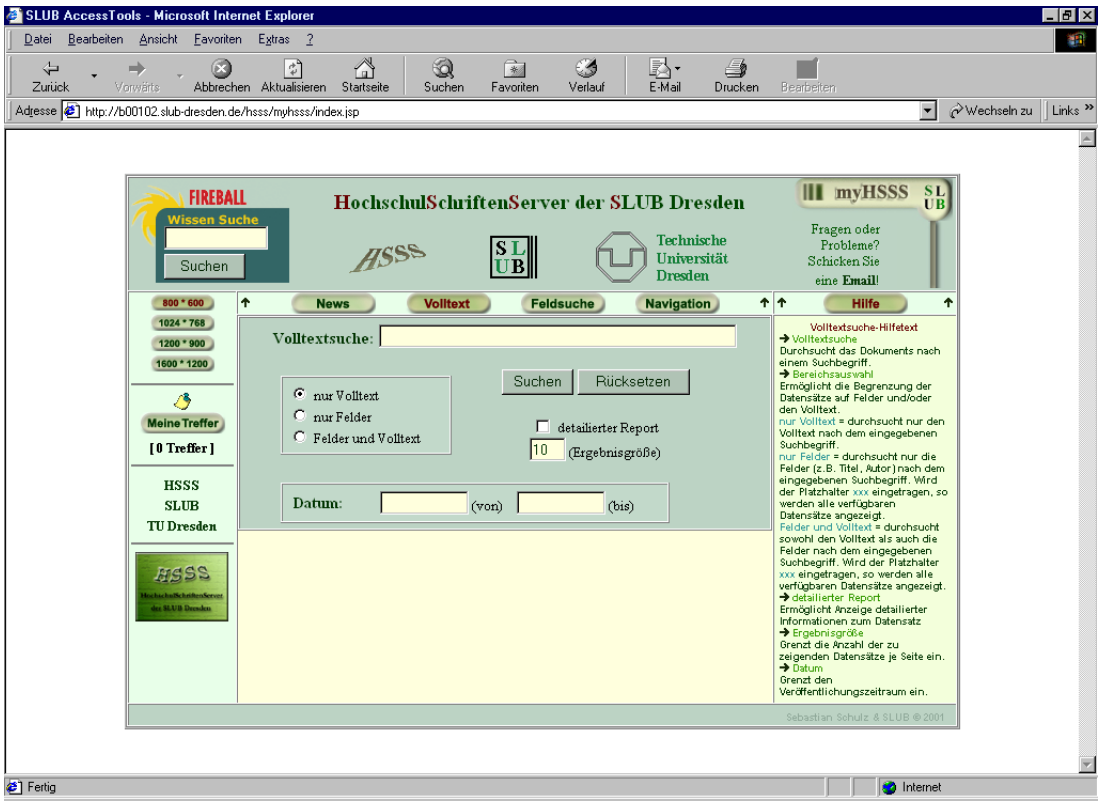

Abbildung 56 *- myHSSS: Volltextsuche in der Auflösung 800*600*

Im oberen Teil befinden sich einige Grafiken, die auf die beteiligten Institutionen verweisen sowie die integrierte Suche von *„Fireball Wissen“*. Da es sich gerade bei dieser dritten Teilanwendung um eine prototypische Implementierung handelt, sind die Inhalte später zu ersetzen bzw. zu ergänzen. Links darunter findet sich eine vertikale Leiste mit einigen Verweisen, Grafiken und Buttons, darunter auch diejenigen,

welche die Wahl von verschiedenen Skalierungen (Auflösungen) ermöglichen. Daneben befindet sich ein relativ großer Bereich, der für die eigentlichen Benutzerschnittstellen reserviert ist (z. Bsp. Volltextsuche, Feldsuche etc.), darüber eine horizontale Leiste mit den verschiedenen Such- bzw. Navigationsmöglichkeiten. Rechts befindet sich ein Bereich, der für die Darstellung der kontextsensitiven Hilfe benutzt wird, die somit für den Nutzer immer verfügbar ist. Standardmäßig startet *myHSSS* in der Auflösung 600*600, wie in **Abbildung 56** zu sehen ist. Durch das Klicken auf einen der vier Auflösungs-Buttons ist vom Nutzer aber jederzeit eine Änderung der Skalierung in den vorgegebenen Bereichen realisierbar. In **Abbildung 57** wurde die Auflösung beispielsweise auf 1024*768 Bildpunkte gesetzt:

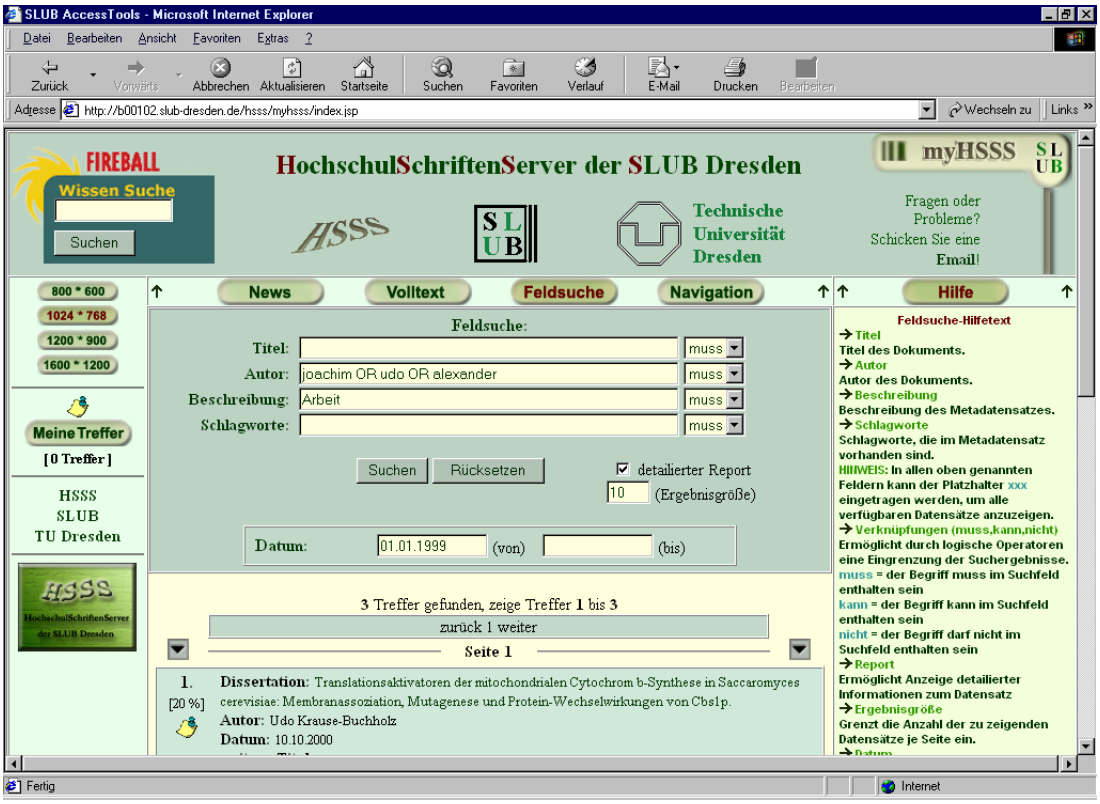

Abbildung 57 - *myHSSS: Feldsuche in der Auflösung 1024*768*

Dabei ist zu anzumerken, dass sich die Skalierung sowohl auf die Maße der einzelnen HTML-Strukturen bezieht (Tabellenbreiten, -höhen etc.) als auch auf Schriftgrößen[26] und Grafiken. Allerdings bestand in der Skalierung der Grafiken das größte Problem, da Vectorgrafikformate von den Browsern zur Zeit leider noch nicht ausreichend

[26]Für die Skalierung von Schriftgrößen wurde allerdings im Gegensatz zur normalen Skalierung ein anderer, nichtlinearer Faktor gewählt (degressives Wachstum).

bzw. gar nicht unterstützt werden und die Skalierung von Bitmap-Grafiken durch den Browser qualitativ nicht befriedigt. So musste ein Ausweg gefunden werden, der darin bestand, die Grafiken mit einem Grafikbearbeitungsprogramm für die gewünschten Auflösungen explizit zu skalieren und unter verschiedenen Namen zu speichern, die dann von der Skalierungsfunktion berechnet werden, so dass für jede Auflösung immer die richtigen Grafiken eingebunden werden. Dies bedeutete zum einen natürlich einen relativ großen Aufwand, da jede Grafik vorher für vier Auflösungen skaliert werden musste, und zum anderen den Verlust der Möglichkeit, übergangslos zu skalieren. Zusätzlich bedeuten die redundanten, explizit skalierten Bilder einen Performanceverlust beim Nachladen einer neuen Auflösung. Trotzdem konnte die dritte Architekturvision, bis auf die genannten Mängel, zufriedenstellend prototypisch realisiert werden, so dass eine optimale Platznutzung in den vier vorgegebenen Skalierungen möglich ist. Daneben wurde ein zusätzliches Konzept realisiert, welches erlaubt, einzelne Teile der Schnittstelle auch in einem anderen Zusammenhang zu nutzen bzw. gesondert darzustellen. Für das zur Feldsuche dienende Formular, welches in **Abbildung 57** normal integriert gezeigt ist, ist auch eine externe Darstellung realisierbar, die nur dieses Formular zeigt, wie in **Abbildung 58** zu sehen ist:

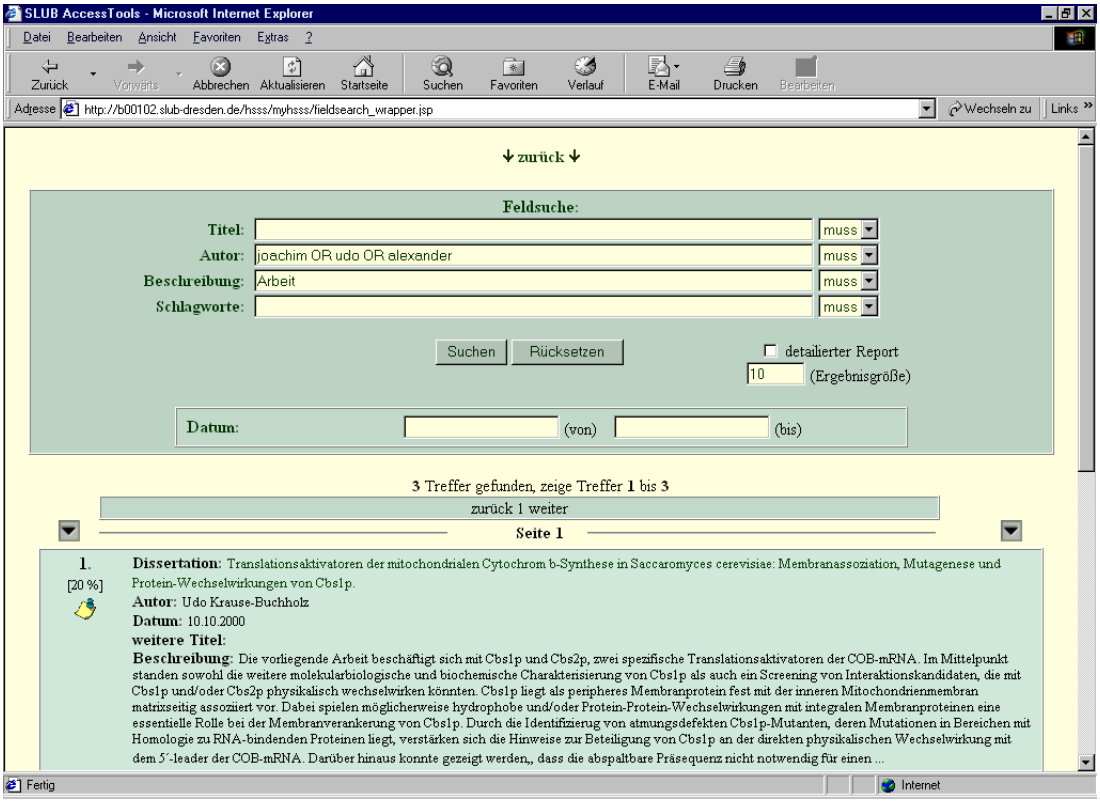
SLUB AccessTools - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://b00102.slub-dresden.de/hsss/myhsss/fieldsearch_wrapper.jsp
Wechseln zu Links
zurück
Feldsuche:
Titel:
Autor: joachim OR udo OR alexander
Beschreibung: Arbeit
Schlagworte:
muss
Suchen Rücksetzen
detailierter Report
10 (Ergebnisgröße)
Datum: (von) (bis)
3 Treffer gefunden, zeige Treffer 1 bis 3
zurück 1 weiter
Seite 1
1. [20 %]
Dissertation: Translationsaktivatoren der mitochondrialen Cytochrom b-Synthese in Saccaromyces cerevisiae: Membranassoziation, Mutagenese und Protein-Wechselwirkungen von Cbs1p.
Autor: Udo Krause-Buchholz
Datum: 10.10.2000
weitere Titel:
Beschreibung: Die vorliegende Arbeit beschäftigt sich mit Cbs1p und Cbs2p, zwei spezifische Translationsaktivatoren der COB-mRNA. Im Mittelpunkt standen sowohl die weitere molekularbiologische und biochemische Charakterisierung von Cbs1p als auch ein Screening von Interaktionskandidaten, die mit Cbs1p und/oder Cbs2p physikalisch wechselwirken könnten. Cbs1p liegt als peripheres Membranprotein fest mit der inneren Mitochondrienmembran matrixseitig assoziiert vor. Dabei spielen möglicherweise hydrophobe und/oder Protein-Protein-Wechselwirkungen mit integralen Membranproteinen eine essentielle Rolle bei der Membranverankerung von Cbs1p. Durch die Identifizierug von atmungsdefekten Cbs1p-Mutanten, deren Mutationen in Bereichen mit Homologie zu RNA-bindenden Proteinen liegt, verstärken sich die Hinweise zur Beteiligung von Cbs1p an der direkten physikalischen Wechselwirkung mit dem 5'-leader der COB-mRNA. Darüber hinaus konnte gezeigt werden,, dass die abspaltbare Präsequenz nicht notwendig für einen ...
Fertig
Internet

Abbildung 58 *- myHSSS: Feldsuche in der Einzel-Ansicht*

Die verschiedenen Formulare für die Suchen und die kontextsensitive Hilfe sind jeweils als separate JSP's realisiert, die in andere JSP's unter bestimmten Voraussetzungen integriert werden können. So lassen sich Teile von *myHSSS* in verschiedenen Kontexten darstellen, die integrierten JSP's sind sozusagen „*View*-Komponenten", die sich relativ flexibel wiederverwenden lassen. Die in **Abbildung 58** gezeigte alleinige Integration der Feldsuche wird durch das Klicken auf die aufwärtsgerichteten Pfeile in der horizontalen Befehlsleiste des *myHSSS* (siehe **Abbildung 57**) erreicht. Will man wieder zur integrierten Ansicht von *myHSSS* zurück, klickt man auf „zurück" bzw. auf die nach unten gerichteten Pfeile. So kann man zwischen einer alles umfassenden Gesamtansicht und einer auf das Wesentliche reduzierten, speziell auf die augenblickliche Aktion konzentrierten Ansicht hin und her wechseln. Dies ist insbesondere bei kleinen Bildschirmauflösungen von Vorteil und kann besonders am Beispiel der kontextsensitiven Hilfe demonstriert werden. Im Vergleich zu der in **Abbildung 57** gezeigten Ansicht der Hilfe, der nicht viel Platz zur Verfügung steht und die nur für einen schnellen Überblick genutzt werden sollte, ist die in **Abbildung 59** allein präsentierte Hilfe aufgrund der geänderten Platzverhältnisse wesentlich übersichtlicher und könnte auch für umfangreichere Nachforschungen genutzt werden:

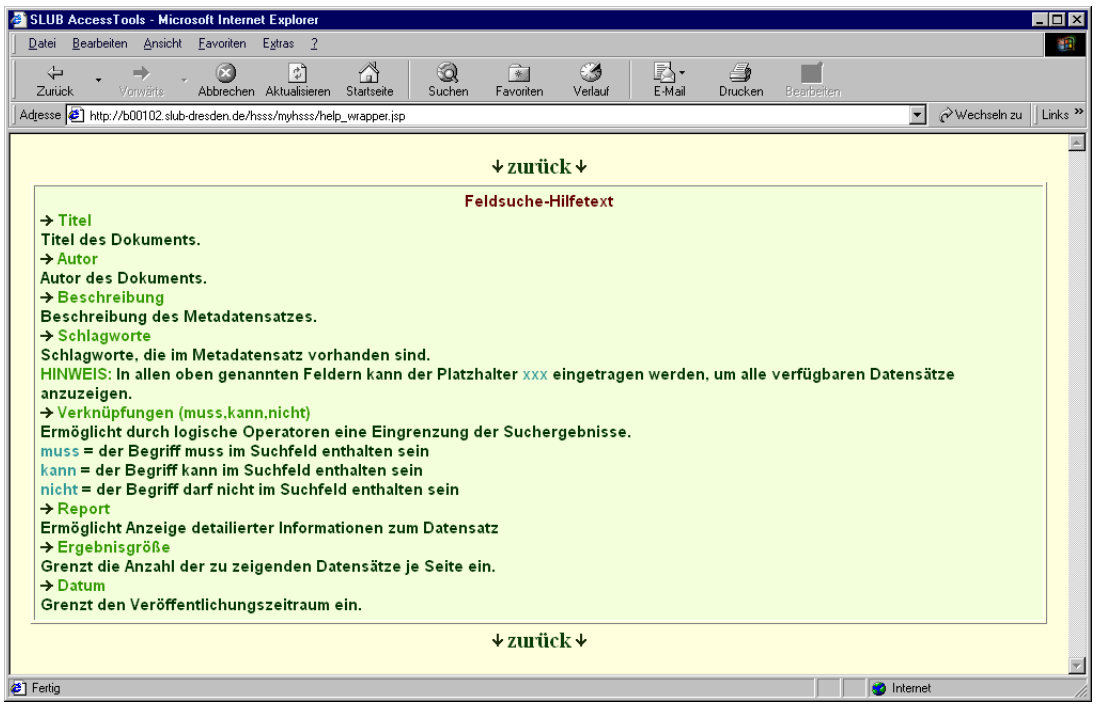
SLUB AccessTools - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://b00102.slub-dresden.de/hsss/myhsss/help_wrapper.jsp
Wechseln zu Links
↓zurück↓
Feldsuche-Hilfetext
→ Titel
Titel des Dokuments.
→ Autor
Autor des Dokuments.
→ Beschreibung
Beschreibung des Metadatensatzes.
→ Schlagworte
Schlagworte, die im Metadatensatz vorhanden sind.
HINWEIS: In allen oben genannten Feldern kann der Platzhalter xxx eingetragen werden, um alle verfügbaren Datensätze anzuzeigen.
→ Verknüpfungen (muss,kann,nicht)
Ermöglicht durch logische Operatoren eine Eingrenzung der Suchergebnisse.
muss = der Begriff muss im Suchfeld enthalten sein
kann = der Begriff kann im Suchfeld enthalten sein
nicht = der Begriff darf nicht im Suchfeld enthalten sein
→ Report
Ermöglicht Anzeige detailierter Informationen zum Datensatz
→ Ergebnisgröße
Grenzt die Anzahl der zu zeigenden Datensätze je Seite ein.
→ Datum
Grenzt den Veröffentlichungszeitraum ein.
↓zurück↓
Fertig
Internet

Abbildung 59 *- myHSSS: Hilfe in der Einzel-Ansicht*

Die Navigation zwischen den beiden Ansichten erfolgt wieder über die aufwärtsgerichteten Pfeile (diesmal bei dem Button „Hilfe", siehe z. Bsp. **Abbildung 57**) respektive durch „zurück" bzw. den abwärtsgereichteten Pfeilen in **Abbildung 59**.

Die vierte Architekturvision, welche eine Portallösung, die Integration externer Dienste und eine weitgehende Personalisierung verlangte, wurde schon in Abschnitt 3.2.2 dahingehend gemildert, dass lediglich eine Session-basierte Umsetzung realisiert werden soll, welche aber die sich daraus ergebenden Möglichkeiten andeutet. Im Rahmen der Diplomarbeit wurde eine interessante Funktionalität umgesetzt, die es erlaubt, einzelne Suchergebnisse (auch aus verschiedenen Suchen wie etwa Volltextsuche oder Feldsuche) zu „sammeln“. Das bedeutet, dass diese Treffer innerhalb der Session in einer bestimmten Datenstruktur gespeichert werden und zu einem beliebigen Zeitpunkt verfügbar sind. Die Treffer werden „gesammelt“, indem der Nutzer auf das „Notizzettel“-Symbol klickt, welches für jeden Treffer generiert wird (siehe z. Bsp. **Abbildung 58**). Dadurch wird der zugehörige Treffer automatisch gespeichert und die Anzeige „Meine Treffer“ aktualisiert (siehe **Abbildung 57** in der linken Spalte). Die gesammelten Treffer können jederzeit angezeigt werden, indem auf den Button „Meine Treffer“ oder auf das „Notizzettel“-Symbol in der linken Spalte von *myHSSS* geklickt wird. Es erscheint eine Ansicht, die der in **Abbildung 60** gezeigten ähnelt:

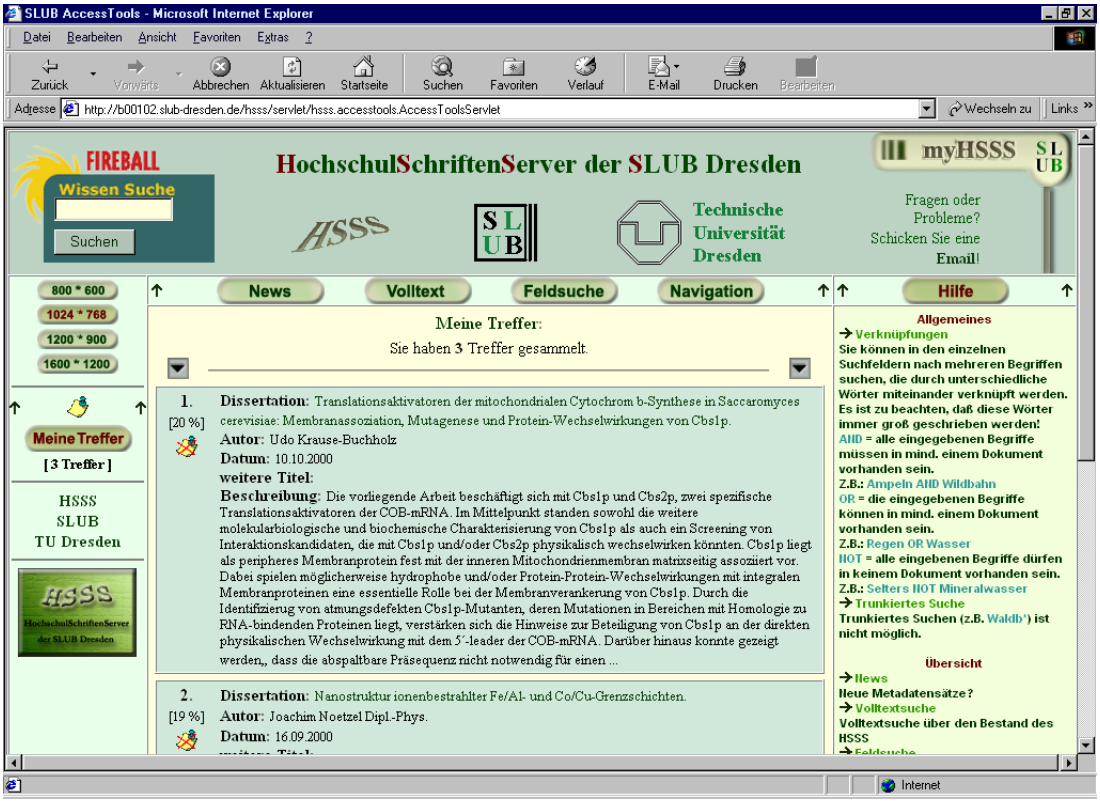
SLUB AccessTools - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://b00102.slub-dresden.de/hsss/servlet/hsss.accesstools.AccessToolsServlet
Wechseln zu Links
FIREBALL
Wissen Suche
Suchen
HochschulSchriftenServer der SLUB Dresden
myHSSS
Technische Universität Dresden
Fragen oder Probleme? Schicken Sie eine Email!
800 * 600
1024 * 768
1200 * 900
1600 * 1200
Meine Treffer
[3 Treffer]
HSSS
SLUB
TU Dresden
News
Volltext
Feldsuche
Navigation
Hilfe
Meine Treffer:
Sie haben 3 Treffer gesammelt.
1. [20 %] Dissertation: Translationsaktivatoren der mitochondrialen Cytochrom b-Synthese in Saccaromyces cerevisiae: Membranassoziation, Mutagenese und Protein-Wechselwirkungen von Cbs1p.
Autor: Udo Krause-Buchholz
Datum: 10.10.2000
weitere Titel:
Beschreibung: Die vorliegende Arbeit beschäftigt sich mit Cbs1p und Cbs2p, zwei spezifische Translationsaktivatoren der COB-mRNA. Im Mittelpunkt standen sowohl die weitere molekularbiologische und biochemische Charakterisierung von Cbs1p als auch ein Screening von Interaktionskandidaten, die mit Cbs1p und/oder Cbs2p physikalisch wechselwirken könnten. Cbs1p liegt als peripheres Membranprotein fest mit der inneren Mitochondrienmembran matrixseitig assoziiert vor. Dabei spielen möglicherweise hydrophobe und/oder Protein-Protein-Wechselwirkungen mit integralen Membranproteinen eine essentielle Rolle bei der Membranverankerung von Cbs1p. Durch die Identifizierug von atmungsdefekten Cbs1p-Mutanten, deren Mutationen in Bereichen mit Homologie zu RNA-bindenden Proteinen liegt, verstärken sich die Hinweise zur Beteiligung von Cbs1p an der direkten physikalischen Wechselwirkung mit dem 5'-leader der COB-mRNA. Darüber hinaus konnte gezeigt werden,, dass die abspaltbare Präsequenz nicht notwendig für einen ...
2. [19 %] Dissertation: Nanostruktur ionenbestrahlter Fe/Al- und Co/Cu-Grenzschichten.
Autor: Joachim Noetzel Dipl.-Phys.
Datum: 16.09.2000
Allgemeines
Verknüpfungen
Sie können in den einzelnen Suchfeldern nach mehreren Begriffen suchen, die durch unterschiedliche Wörter miteinander verknüpft werden. Es ist zu beachten, daß diese Wörter immer groß geschrieben werden!
AND = alle eingegebenen Begriffe müssen in mind. einem Dokument vorhanden sein.
Z.B.: Ampeln AND Wildbahn
OR = die eingegebenen Begriffe können in mind. einem Dokument vorhanden sein.
Z.B.: Regen OR Wasser
NOT = alle eingebenen Begriffe dürfen in keinem Dokument vorhanden sein.
Z.B.: Selters NOT Mineralwasser
Trunkiertes Suche
Trunkiertes Suchen (z.B. Waldb*) ist nicht möglich.
Übersicht
News
Neue Metadatensätze?
Volltextsuche
Volltextsuche über den Bestand des HSSS
Internet

Abbildung 60 *- myHSSS: gesammelte Treffer*

Hier werden alle gesammelten Treffer in einer ähnlichen Form wie bei den verschiedenen Suchen präsentiert. Zu beachten ist dabei, dass sie zur besseren Übersicht in der

Reihenfolge angezeigt werden, in der sie vom Nutzer „gesammelt“ wurden. Anhand der absoluten URL eines Treffers sind Treffer von der Anwendung eindeutig unterscheidbar, so dass kein Treffer mehrfach aufgeführt wird, was schnell passieren könnte, wenn ein Nutzer in verschiedenen Suchen denselben Treffer erhält und ihn mehrmals zum „Sammeln“ anklickt. Allerdings würde sich ein solcher, mehrfach gewählter Treffer bei jedem neuen Klick in seiner Positionierung innerhalb der Trefferliste verändern[27]. Diese zeitliche Positionierung wird durch die in Abschnitt 6.2.2 vorgestellte Klasse `TimeHit` ermöglicht. Ein Klick auf das Symbol des durchgestrichenen Notizzettels, welches zu jedem gesammelten Treffer angezeigt wird, entfernt diesen wieder aus der Trefferliste. Die integrierte Komponente der gesammelten Treffer kann ebenso wie die Komponenten für die Suchen und die Hilfe in einer Einzel-Ansicht gezeigt werden, welche durch das Klicken der aufwärtsgerichteten Pfeile bei „Meine Treffer“ erreicht wird. Im Folgenden soll noch einmal die Trefferanzeige bei Suchen näher erläutert werden, dazu dient die in **Abbildung 61** ausschnittsweise gezeigte Ergebnismenge einer Volltextsuche:

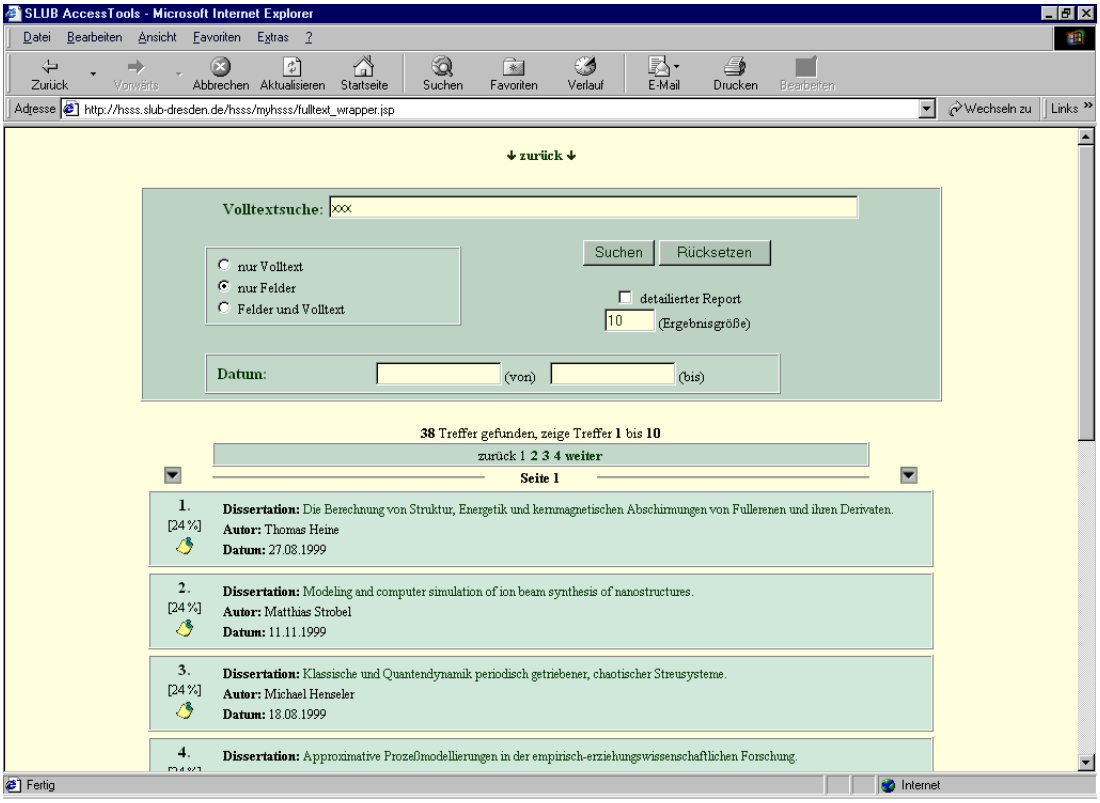
SLUB AccessTools - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://hsss.slub-dresden.de/hsss/myhsss/fulltext_wrapper.jsp
Wechseln zu Links
zurück
Volltextsuche: xxx
Suchen Rücksetzen
nur Volltext
nur Felder
Felder und Volltext
detailierter Report
10 (Ergebnisgröße)
Datum: (von) (bis)
38 Treffer gefunden, zeige Treffer 1 bis 10
zurück 1 2 3 4 weiter
Seite 1
1. [24 %] Dissertation: Die Berechnung von Struktur, Energetik und kernmagnetischen Abschirmungen von Fullerenen und ihren Derivaten.
Autor: Thomas Heine
Datum: 27.08.1999
2. [24 %] Dissertation: Modeling and computer simulation of ion beam synthesis of nanostructures.
Autor: Matthias Strobel
Datum: 11.11.1999
3. [24 %] Dissertation: Klassische und Quantendynamik periodisch getriebener, chaotischer Streusysteme.
Autor: Michael Henseler
Datum: 18.08.1999
4. Dissertation: Approximative Prozeßmodellierungen in der empirisch-erziehungswissenschaftlichen Forschung.
Fertig
Internet

Abbildung 61 *- myHSSS: Teilergebnis einer Volltextsuche*

[27] ganz nach unten rutschen

Wie in Abschnitt 6.2.2 bereits angedeutet, wird das Gesamtergebnis auf mehrere Teilergebnisse (Seiten) aufgeteilt, wenn es eine bestimmte Trefferzahl übersteigt[28]. Oberhalb und unterhalb der jeweils angezeigten Treffer befindet sich deshalb eine Navigationsleiste, welche die gezielte Anzeige bestimmter Teilergebnisse bzw. Seiten durch das Klicken auf „zurück“, „weiter“ oder die jeweilige Seitennummer ermöglicht. Außerdem werden die Gesamttrefferzahl genannt sowie die Nummern der innerhalb der aktuellen Seite angezeigten Treffer. Jeder Treffer wird innerhalb eines eigenen, visuell abgegrenzten Bereichs präsentiert. Auf der linken Seite befinden sich jeweils die Statusinformationen eines Treffers, so zum Beispiel dessen Nummer innerhalb des Gesamtergebnisses und dessen Wertigkeit[29]. Darunter befindet sich das Notizzettel-Symbol, welches beim Anklicken zur Speicherung des jeweiligen Treffers in der Menge der gesammelten Treffer führt.

Auf der rechten Seite jedes Trefferbereichs werden die Informationen zu der entsprechenden Hochschulschrift in Abhängigkeit des gewählten Detaillierungsgrades präsentiert. Dies sind vor allem der Dokumententyp und der Originaltitel der Hochschulschrift. Dieser ist als Link realisiert, der beim Aktivieren die passende Frontdoor in einem neuen Fenster öffnet. Außerdem werden die Autoren und das Veröffentlichungsdatum der Hochschulschrift genannt. Ist die Option „detaillierter Report“ vom Benutzer aktiviert worden, werden zusätzlich alle weiteren Titel (alternative oder übersetzte Titel) und die vorhandenen Beschreibungen (Abstracts) für die Hochschulschrift angezeigt.

Zum Abschluss sollen noch einmal die bisher innerhalb von *myHSSS* verfügbaren Funktionen zusammengefasst und kurz erläutert werden:

- **News**
  Hier lassen sich alle innerhalb eines bestimmten Zeitraums veröffentlichten Hochschulschriften anzeigen. Die vorgegebenen Rubriken sind „1 Woche“, „2 Wochen“, „1 Monat“, „3 Monate“ und „6 Monate“, jeweils zurückgerechnet vom aktuellen Datum.

- **Volltextsuche**
  Die Volltextsuche ermöglicht eine Suche innerhalb des Volltextes und/oder (relativ grob) in allen Feldern des Index einer Hochschulschrift, ohne dass diese

[28] die vom Nutzer durch eine entsprechende Eingabe bei „Ergebnisgröße“ festgesetzt werden kann

[29] Diese Wertigkeit wird von *Lucene* berechnet (scoring) und hängt u.a. von der Varianz des Suchbegriffs in den zu durchsuchenden Feldern ab. Der Rang bzw. die Nummer des Treffers innerhalb der Ergebnismenge steht in direktem Zusammenhang mit diesem Wert.

Felder näher benannt sind[30]. Dadurch ergeben sich innerhalb der Volltextsuche die folgenden Suchmodi[31]:

- *nur Volltext*
  Es wird nur der Volltext der Hochschulschrift durchsucht (Feld „`fulltext`“ innerhalb der *Lucene*-Dokumente)
- *nur Felder*
  Es werden alle indizierten Felder durchsucht, mit Ausnahme des Volltext-Feldes („`fulltext`“)
- *Felder und Volltext*
  Es werden alle indizierten Felder von *Lucene* durchsucht

- **Feldsuche**
  Die Feldsuche erlaubt die fein abgestimmte Suche in bestimmten Feldern des Index. Es lassen sich separate Anfragen für die Felder
  - *Titel* (*Lucene*-Dokumentenfeld „`title`“)
  - *Autor* (*Lucene*-Dokumentenfeld „`creator`“)
  - *Beschreibung* (*Lucene*-Dokumentenfeld „`description`“)
  - *Schlagworte*[32] (*Lucene*-Dokumentenfeld „`subject`“)

  bilden und durch die Konjunktionen **muss**, **kann** oder **nicht** beliebig miteinander verknüpfen (siehe auch Abschnitt 6.2.1). Dadurch können hochkomplexe Gesamtanfragen entstehen.

- **Navigation**
  Die Navigation ist sehr einfach implementiert und erlaubt, alle Treffer für bestimmte Kategorien anzuzeigen. Diese Kategorien sind:
  - *alle Veröffentlichungsjahre*
    z. Bsp. alle Hochschulschriften des Jahres 2000
  - *alle Fakultäten*
    z. Bsp. alle Hochschulschriften der Fakultät „Wirtschaftswissenschaften“
  - *alle Dokumententypen*
    z. Bsp. alle Hochschulschriften der Dokumentenart „Habilitation“

- **Meine Treffer**
  Ermöglicht die Betrachtung aller während einer Sitzung „gesammelten“ Treffer sowie deren Löschung.

[30] obwohl dies explizit innerhalb der Suchanfrage möglich ist (siehe dazu die Syntax von Suchanfragen in Abschnitt 6.2.1)
[31] diese Suchen gehorchen einer internen Oder-Semantik
[32] auch Klassifikationen, damit ergibt sich eine interessante Suchmöglichkeit für einen Bibliothekar

6.4. Einschätzung

Bei der dritten Teilanwendung handelt es sich um die „jüngste“ der drei Teilanwendungen des HSSS, die erst vollständig während der Diplomarbeit konzipiert und entwickelt wurde. Aus diesem Grund treffen auf sie die Merkmale einer prototypischen Realisierung im Besonderen zu. Bevor dieser Abschnitt auf die zahlreichen Verbesserungen und Erweiterungen eingeht, die hier möglich sind, sollen die positiven Eigenschaften der dritten Teilanwendung gewürdigt werden.

So konnte die dritte Architekturvision recht überzeugend umgesetzt werden, wenngleich sich die bereits angedeuteten Schwierigkeiten bei der Skalierung von Grafiken ergaben. Hier ist eine wirklich nützliche Funktionalität realisiert worden, die als Beispiel für die Gestaltung variabler Benutzerschnittstellen auch für zukünftige Weiterentwicklungen an den beiden anderen Teilanwendungen dienen kann. Auch der Umfang der realisierten Such- bzw. Navigationsmöglichkeiten ist angemessen und entspricht dem gängiger Suchmaschinen. Insbesondere die Realisierung der Suchen und die Präsentation von Ergebnissen sowie die Unterstützung von Teilergebnissen (Seiten) und deren Verwaltung kann schon in dieser prototypischen Version überzeugen. Die Verwendung von *Lucene* hat sich bis jetzt wirklich ausgezahlt, eine direkte Realisierung der Suchen innerhalb der Datenbank wäre bedeutend aufwendiger und mit Sicherheit weniger performant gewesen. Die Geschwindigkeit der Suchen ist subjektiv sehr hoch, hier sollten allerdings noch Messungen bei massiv-parallelem Zugriff und bei größeren Datenmengen vorgenommen werden[33].

Die vierte Architekturvision konnte nur in Ansätzen realisiert werden, was allerdings bereits in Abschnitt 3.2.2 diskutiert und erwartet wurde. Immerhin wurde ein auf ein Portal ausgerichtetes prototypisches Design für die Benutzeroberfläche *myHSSS* realisiert, welches freilich einige inhaltliche Lücken aufweist, die im Rahmen der Weiterentwicklung des HSSS zu schließen sind. Ansätze für mögliche Dienste wurden mit der Möglichkeit aufgezeigt, Treffer beliebiger Suchen innerhalb einer Sitzung zu sammeln. Solche und viele weitere Dienste, insbesondere Personalisierungen, ließen sich bei einer Erweiterung der dritten Teilanwendung auf persistente Registrierung etc. leicht realisieren. Zusammenfassend lässt sich sagen, dass die entstandenen Versionen von *myHSSS* und *AccessTools* durchaus dazu geeignet sind, vorläufig den Zugriff auf alle Hochschulschriften des Gesamtsystems in Form von Suche bzw. Navigation sicherzustellen, allerdings sollten gerade sie kontinuierlich weiterentwickelt und verbessert

[33] wobei auch Testsuchen in ca. 40 Volltexten mit durchschnittlich 200 KByte Text nach einem Stopwortlisten-Wort wie „so“ ausgezeichnete Antwortzeiten lieferten

werden. Eine Reihe nützlicher Hinweise auf solche Erweiterungen und Verbesserungen soll der letzte Teil dieses Kapitels liefern (diese Verbesserungen werden in Abschnitt 8.2 ab Seite 222 noch einmal strukturiert zusammengefasst).

- Der gegenwärtig verwendete Abbildungsansatz der expliziten Abbildung ist aus den in Abschnitt 6.1 genannten Gründen zu überdenken und gegebenenfalls durch den Ansatz einer impliziten Abbildung mit Hilfe einer Abbildungsfunktion zu ersetzen. Dabei sollten Testmessungen belegen, ob ein solcher Wechsel wirklich sinnvoll und notwendig ist. Bei einer Umstellung ist darauf zu achten, dass bereits veröffentlichte absolute URL's weiterhin korrekt abgebildet werden müssen, so dass wahrscheinlich zwei verschiedene Abbildungen unterstützt werden müssen.

- Die gegenwärtig genutzte Stopwortliste (realisiert innerhalb der Klasse `HsssStopAnalyzer`) ist zu überprüfen und zu vervollständigen. Außerdem wäre die Implementation eines deutschen Stemming-Algorithmus und dessen Integration in Lucene eine wirksame Erweiterung und eine lohnende Aufgabe[34].

- Für die bisher sehr aufwendige Skalierung der Grafiken durch vorherige Erstellung von separaten Grafiken für jede unterstützte Auflösung sollten andere Lösungen gefunden werden. Hier besteht allerdings eine Abhängigkeit von der Browserentwicklung und insbesondere deren Fähigkeiten in Bezug auf Vectorgrafikformate[35]. Gegebenenfalls ist auch die Verfügbarkeit einer alternativen Version von *myHSSS* ohne Grafiken sinnvoll.

- Die relativ einfache Realisierung der Navigation sollte deutlich erweitert werden. So wäre eine grafisch intuitive Präsentation der Navigation in Form einer Baumstruktur o.ä. sinnvoll. Eine mögliche Realisierung der Navigation wäre die, sie so unabhängig wie möglich von den schon getroffenen Einschränkungen zu machen, so dass jederzeit noch alle sinnvollen Navigationsmöglichkeiten offen stehen[36].

- Die Benutzeroberfläche *myHSSS* sollte kontinuierlich zu einem Portal ausgebaut werden, so dass es der einzige zentrale Interaktionspunkt für alle im Zusammenhang mit dem HSSS stehenden Informationen und Funktionalitäten für alle

[34] Eine solche Implementierung sollte selbstverständlich veröffentlicht und anderen Nutzern, insbesondere den Entwicklern von *Lucene*, zugänglich gemacht werden.

[35] Hier sollte vor allem die Entwicklung XML-basierter Formate wie SVG und deren Unterstützung verfolgt werden.

[36] Damit ist die Bildung beliebiger erreichbarer Teilmengen gemeint, so dass z. Bsp. die durch die Wahl von {`Veröffentlichungsjahr=2000`} entstehende Teilmenge durch {`Fakultät=Informatik`}, aber auch durch {`Dokumententyp=Forschungsbericht`} wahlfrei verkleinerbar ist, genauso gut aber auch die Reihenfolge {`Dokumententyp=Forschungsbericht`} und anschließend {`Veröffentlichungsjahr=2000`} gewählt werden dürfte (und zum selben Ergebnis führt)

Nutzer ist[37]. Ein solches Portal sollte zusätzlich externe Inhalte integrieren[38] (z. Bsp. solche anderer bibliothekarischer Institutionen etc.). Außerdem ist eine datenbankgestützte Realisierung von persistenten Logins notwendig, um ein dauerhaft personalisiertes Angebot etablieren zu können. Daraus ergeben sich weitere vielfältige Dienste, wie etwa eine persönliche, abspeicherbare Sicht der Benutzeroberfläche (Farben, Anordnungen[39] etc.) oder persistent abspeicherbare Treffersammlungen. Außerdem wäre die Speicherung von Interessensprofilen denkbar, welche die automatische Benachrichtigung der Nutzer im Falle passender, neu eintreffender Hochschulschriften ermöglichen würde und vieles mehr. Die technischen Voraussetzungen für die Realisierung solcher Angebote existieren bereits jetzt.

[37] Das bedeutet, dass die bisherigen HSSS-Seiten hier integriert werden sollten

[38] wie mit der Integration der „Fireball Wissen“-Suche bereits angedeutet

[39] dazu könnte der gezeigte *View*-Komponenten-Ansatz erweitert und ausgebaut werden

7. Implementierung des OAI-Protokolls

Inhalt des Kapitels:

7.1. Objektorientierte Analyse und Design 210
7.2. Realisierung 215
7.3. Einschätzung 218

In diesem Kapitel soll die Implementierung des OAI-Protokolls und deren Entwurf für den HSSS erläutert werden. Da auf das OAI-Protokoll bereits in Abschnitt 2.4.1.1 auf Seite 40 eingegangen wurde und mit [L-OAc] eine detaillierte Protokollspezifikation existiert, kann dieses Kapitel relativ kurz gefasst werden und dabei vor allem die Besonderheiten der Implementierung fokussieren.

7.1. Objektorientierte Analyse und Design

Alle OAI-spezifischen Klassen wurden in dem separaten Package „`hsss.oai`“ zusammengefasst. **Abbildung 62** zeigt die Klassen dieses Packages in einem Klassendiagramm:

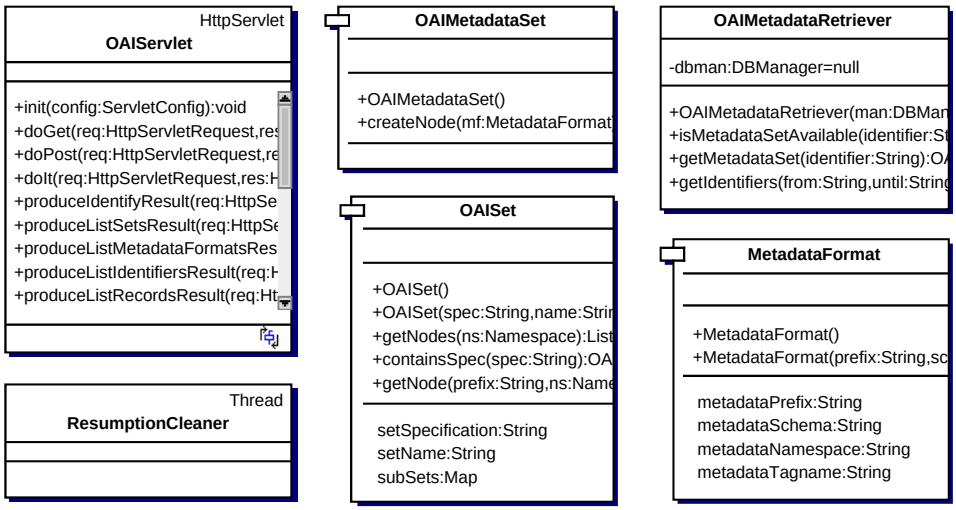

Abbildung 62 *- HSSS Package "oai"*

Da das OAI-Protokoll HTTP als Trägerprotokoll benutzt, wird eine Anwendung benötigt, die sowohl HTTP als auch die Spezifika des OAI-Protokolls versteht. Innerhalb des HSSS-Gesamtsystems wurde diese Anwendung selbstverständlich in Form eines Servlets realisiert. Die Klasse `OAIServlet`, die von `HttpServlet` erbt, ist also die eigentliche OAI-Schnittstelle des HSSS, die allerdings weitere Klassen integriert, die im Folgenden vorgestellt werden. Alle wesentlichen Funktionalitäten, wie etwa das Parsen der OAI-Befehle[1] bzw. der weiteren Parameter sowie das Initiieren der ent-

[1] gekennzeichnet durch den ausgezeichneten Parameter „verb“

sprechenden Reaktionen[2], werden aber bereits durch das `OAIServlet` realisiert. Die meisten OAI-spezifischen Archivinformationen[3] werden dem Servlet dabei über die `web.xml` bekannt gemacht, so dass diese Informationen ohne Änderung des Quellcodes konfigurierbar sind.

Die eigentlichen Informationen zu den Hochschulschriften des Archivs befinden sich jedoch in Form von Metadatensätzen in der Datenbank. Mit Hilfe der Klasse `OAIMetadataRetriever` können diese Informationen dem `OAIServlet` zur Verfügung gestellt werden. `OAIMetadataRetriever` benutzt wiederum den schon in Abschnitt 3.3.4.3 vorgestellten `DBManager` und erstellt als Ergebnis Objekte des Typs `OAIMetadataSet`, welche einen Metadatensatz speziell für die Zwecke des OAI-Protokolls kapseln. Mit Hilfe der Klasse `MetadataFormat`, welche ein OAI-Metadatenformat beschreibt[4] und dafür Attribute wie `metadataPrefix`, `metadataScheme` oder `metadataNamespace` besitzt, kann die Klasse `OAIMetadataSet` die vom OAI-Protokoll geforderte XML-Repräsentation von sich selbst durch die Methode `createNode()` erstellen. Die so entstehenden XML-Strukturen werden dann vom `OAI-Servlet` zu der ebenfalls XML-basierten Gesamtstruktur komponiert, welche innerhalb einer HTTP-Antwort an den Aufrufer zurückgesendet wird.

Die Klasse `OAISet` repräsentiert eine OAI-*Set*-Struktur, welche im Rahmen des OAI-Protokolls optional implementiert werden kann. Ein *Set* fasst dabei eine bestimmte Menge von *Records* zu einer Struktur zusammen, die es somit erlaubt, alle Mitglieder dieser Struktur gemeinsam durch das Protokoll anzusprechen (zu den Definitionen der einzelnen Protokollbegriffe siehe 2.4.1.1). Die Art der Definition solcher *Set*-Strukturen wird von der OAI-Protokoll-Spezifikation nicht vorgeschrieben. Hier hat allerdings die DINI in einem Arbeitspapier[DINa01] vorgeschlagen, für alle deutschen Archive einheitliche *Set*-Definitionen zu nutzen und schlägt vor, folgende formale bzw. inhaltliche *Sets* anzubieten:

1. **Formaler Set**
   Damit sind formale Kriterien gemeint, die den Publikationstyp näher beschreiben, für den HSSS sind das:
   - Dissertation (Dissertation, Habilitation)
   - Bericht (Forschungsbericht, Studie, etc.)

[2]Erstellen der entsprechenden XML-basierten Protokollantworten auf die OAI-Protokollanfragen bzw. der Fehlermeldungen bei fehlenden oder unkorrekten Parametern

[3]etwa Archivname, implementierte Protokollversion, Archiv-Url, Inhaltsbeschreibungen, Beschreibungen der rechtlichen Bedingungen usw.

[4]z. Bsp. *Dublin Core* als das OAI-Pflicht-Metadatenformat

2. **Qualitativer Set**
   Soll später in Anlehnung an sogenannte „Quality-Schemes“ erfolgen, allerdings scheinen die Elemente noch nicht vollständig festgelegt zu sein.

3. **Inhaltlicher Set**
   Die grobe inhaltliche Einordnung soll eine fachliche Selektion über das OAI-Protokoll ermöglichen. DINI schlägt vor, diese Strukturierung den DNB-Sachgruppen nachzuempfinden, z. Bsp. (Ausschnitt):
   - 01 : Wissenschaft und Kultur allgemein
   - 11 : Psychologie
   - 17 : Wirtschaft
   - 28 : Informatik, Datenverarbeitung
   - usw.

Die *Sets* 1 und 3 der Aufzählung werden von der OAI-Implementierung des HSSS unterstützt. Die notwendigen Daten zur Definition der Sets wurden dabei XML-basiert in der Konfigurationsstruktur `oaisets.xml` realisiert, was den großen Vorteil hat, dass sie ohne Eingriff in den Quelltext der Protokoll-Implementation beispielsweise von den Mitarbeitern des Referats 5.3 änderbar sind (Details zu der Konfigurationsstruktur `oaisets.xml` finden sich im Anhang in Abschnitt B). Diese XML-Struktur wird bei der Initialisierung vom `OAIServlet` eingelesen, geparst und in einer Datenstruktur abgelegt, die eine (hierarchische) Menge von Objekten des Typs `OAISet` enthält.

Eine weitere optionale Anforderung des OAI-Protokolls ist die Unterstützung von sogenannten *„Resumtion Token“*. Dabei handelt es sich um einen eindeutigen Wiederaufnahmezeiger, der zusammen mit einer Teilergebnismenge auf eine Anfrage zurückgeschickt wird, die eine Ergebnismenge oberhalb eines zu definiernden Schwellwertes erzeugt[5]. Diese Schwellwerte können im Falle der OAI-Implementation des HSSS von außen über die XML-Datei `web.xml` und darüberhinaus auch für unterschiedliche Befehle[6] eingestellt werden. Die besondere Schwierigkeit besteht aufgrund der Zustandslosigkeit von HTTP darin, Wiederaufnahmezeiger, Datensätze und Anfragen korrekt

[5] Diese Möglichkeit ist aufgrund der Konzeption des OAI-Protokolls und seiner Befehle insbesondere bei großen Archiven sehr sinnvoll. So liefert der OAI-Befehl „ListRecords“ **alle** Metadatesätze eines Archivs, unter Umständen sogar in mehreren Metadatenformaten. Wenn man von einer XML-Struktur ausgeht, die pro Metadatensatz etwa 20 bis 100 Zeilen benötigt und weiterhin ein Archiv mit 5000 Metadatensätzen annimmt, kann man sich das Datenvolumen vorstellen, das in diesem Fall über das Netz geht.

[6] Die OAI-Befehle „ListSets“, „ListIdentifiers“ und „ListRecords“ erlauben die Verwendung von „Resumtion Tokens“ und produzieren Ergebnisse unterschiedlichen Umfangs, weshalb differenzierte Schwellwerte sinnvoll sind.

miteinander in Verbindung zu bringen. Innerhalb der OAI-Implementation des HSSS ist dieses Problem durch das Speichern des Wiederaufnahmezeigers als Schlüssel in einer Hashtabelle gelöst worden. Jedem Wiederaufnahmezeiger ist dort als Wert die noch verbleibende Ergebnismenge zugeordnet. Der Wiederaufnahmezeiger, der automatisch generiert wird, hat dabei die folgende Form:

```
rt<timestamp><4-digit-random>p<count of parts>, z. Bsp. rt9964799925314492p4
```

Die beiden Anfangsbuchstaben sollen den Wiederaufnahmezeiger lediglich optisch von anderen Signaturen des HSSS leichter unterscheidbar machen, der Zeitstempel entspricht der Zahl der Millisekunden seit dem 01. Januar 1970 um 00:00 Uhr GMT, die 4-stellige Zufahlszahl wird in diesem Fall durch den kryptographisch unsicheren Zufallsgenerator im Package `java.util` berechnet[7]. Der Buchstabe „p“ steht für „parts“ und dient als Trennmarkierung der folgenden Ziffern, die angeben, in wieviele Teile das Gesamtergebnis infolge des aktuellen Schwellwertes separiert wurde.

Alle diese Informationen werden von der Klasse `ResumptionCleaner` benötigt, die als einzige Klasse des Klassendiagramms noch nicht betrachtet wurde. Ihre Notwendigkeit ergibt sich aus der Überlegung, dass eine Anfrage, die eine Teilmenge zusammen mit einem Wiederaufnahmezeiger zurückgeliefert hat, vom Aufrufenden nicht notwendigerweise wiederaufgenommen werden muss. Das führt dazu, dass die (unter Umständen sehr große) Restmenge des Ergebnisses umsonst in der dafür vorgesehenen Datenstruktur (zum Beispiel im Hauptspeicher) verbleibt. Vergegenwärtigt man sich zusätzlich, dass es sich bei der Protokoll-Implementierung um eine serverseitige Anwendung handelt, die möglicherweise sehr lange unterbrechungsfrei ausgeführt wird, ist es vorstellbar, dass das beschriebene Verhalten auf Dauer dazu führt, dass wesentliche Teile des zur Verfügung stehenden Speichers vergeudet werden und im schlimmsten Fall sogar die Aufrechterhaltung des Dienstes gefährdet ist.

Deshalb muss es eine Funktion geben, die nach einer bestimmten Zeit die verbliebenen, nicht „abgeholten“ Restmengen aus der entsprechenden Datenstruktur entfernt. Diese Funktion übernimmt die als eigenständiger Thread realisierte Klasse `ResumptionCleaner`, welche gewisse Ähnlichkeiten zu der in Abschnitt 6.1 im Rahmen der URL-Abbildung vorgestellten Klasse `UrlMapper` besitzt. Sie wird bei der Initialisierung der Klasse `OAIServlet` ebenfalls initialisiert und verfügt über eine Referenz auf die Datenstruktur, in der das `OAIServlet` die Assoziationen von Wiederaufnahmezeigern und Restmengen verwaltet. Die Klasse `ResumptionCleaner` untersucht periodisch[8] alle Wiederaufnahmezeiger daraufhin, ob sie noch aktuell sind. Die Ent-

[7] Da dieser Zufallsgenerator wesentlich schneller ist und die kryptographisch sicheren Zufallszahlen an dieser Stelle nicht unbedingt notwendig sind.

[8] Die Periodendauer kann wieder über die Konfigurationsdatei `web.xml` eingestellt werden.

scheidung, ob ein Wiederaufnahmezeiger aktuell ist, wird dabei anhand des folgenden Algorithmus getroffen:

Annahme: es sei *clean-delay* ein Wert in Millisekunden, der angibt, wie alt ein Wiederaufnahmezeiger pro Teilergebnismenge werden darf

Annahme: es sei *hash-map* die Datenstruktur, die die Assoziationen von Wiederaufnahmezeigern und Restmengen enthält

- iteriere über alle Wiederaufnahmezeiger von *hash-map*, führe jeweils folgende Aktionen aus:
  1. hole den nächsten Wiederaufnahmezeiger aus *hash-map*
  2. extrahiere aus dem Wiederaufnahmezeiger den Zeitstempelbereich
     $\Longrightarrow$ *init-timestamp*
  3. extrahiere aus dem Wiederaufnahmezeiger die Anzahl der Teilergebnismengen
     $\Longrightarrow$ *parts*
  4. Berechne das Alter des Wiederaufnahmezeigers durch
     (aktueller Zeitstempel $-$ *init-timestamp*)
     $\Longrightarrow$ *age-timestamp*
  5. Wenn *age-timestamp* größer als (*parts* $*$ *clean-delay*) ist:
     entferne den Wiederaufnahmezeiger und die zugehörige Restmenge aus *hash-map*

Dieser Algorithmus erlaubt ein automatisches Säubern der Datenstruktur für die Wiederaufnahmetoken, so dass immer nur eine begrenzte Zahl aktueller Wiederaufnahmetoken präsent ist. Dabei ist die Aktualisierung der Datenstruktur durch die Thread-basierte Realisierung sogar zur Laufzeit und ohne merkliche Beeinträchtigung des `OAIServlets` möglich[9]. Der Wert für *clean-delay* ist wieder in `web.xml` konfigurierbar.

[9] die Datenstruktur *hash-map* wird durch ein Semaphor geschützt

7.2. Realisierung

Dieser Abschnitt soll sehr kurz gefasst werden und zum besseren Verständnis des Protokolls lediglich dessen Struktur visualisieren. **Abbildung 63** zeigt die Antwort auf eine OAI-„Identify“-Anfrage an die OAI-Implementation des HSSS in einem XML-fähigen Browser:

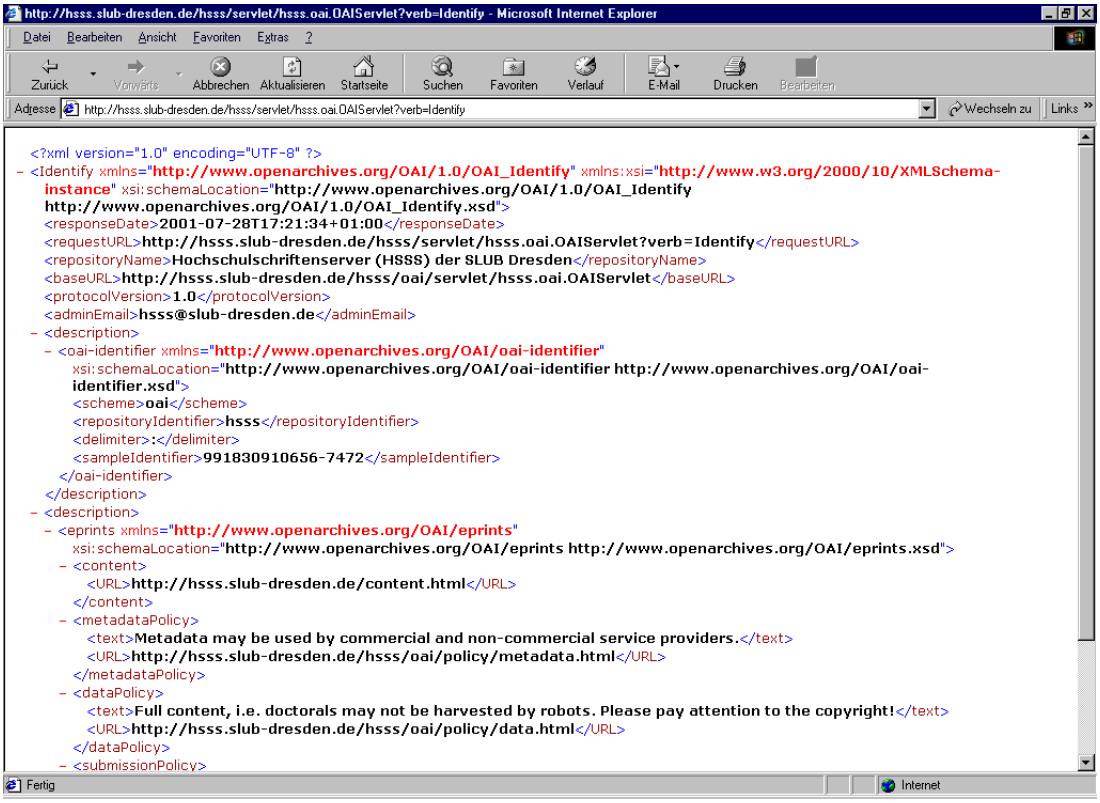

Abbildung 63 *- XMl-Struktur der Antwort auf eine „Identify“-Anfrage*

Wie deutlich zu sehen ist, besteht die Antwort aus einer bestimmten XML-Struktur, die in die HTTP-Antwort eingebettet wurde. Die XML-Struktur ist in einem internationalen Unicode-Format (UTF-8) kodiert, die HTTP-Antwort allerdings in „ISO-8859-1“, um auch deutsche Umlaute darstellen zu können. Die Antwort enthält Informationen über das Archiv wie etwa Archivname (`repositoryName`), die Basis-URL der Anwendung (`baseURL`) oder die implementierte Protokollversion (`protocolVersion`) usw., die einem *Service Provider* wichtige Hinweise liefern, um die Daten des Archivs korrekt auswerten und entsprechende Dienste anbieten zu können. Eine Mimimalimplementation eines solchen Dienstes ist in **Abbildung 64** zu sehen, sie erlaubt es, eine

bestehende OAI-Implementation zu testen und zeigt die Reaktion auf die möglichen OAI-Befehle:

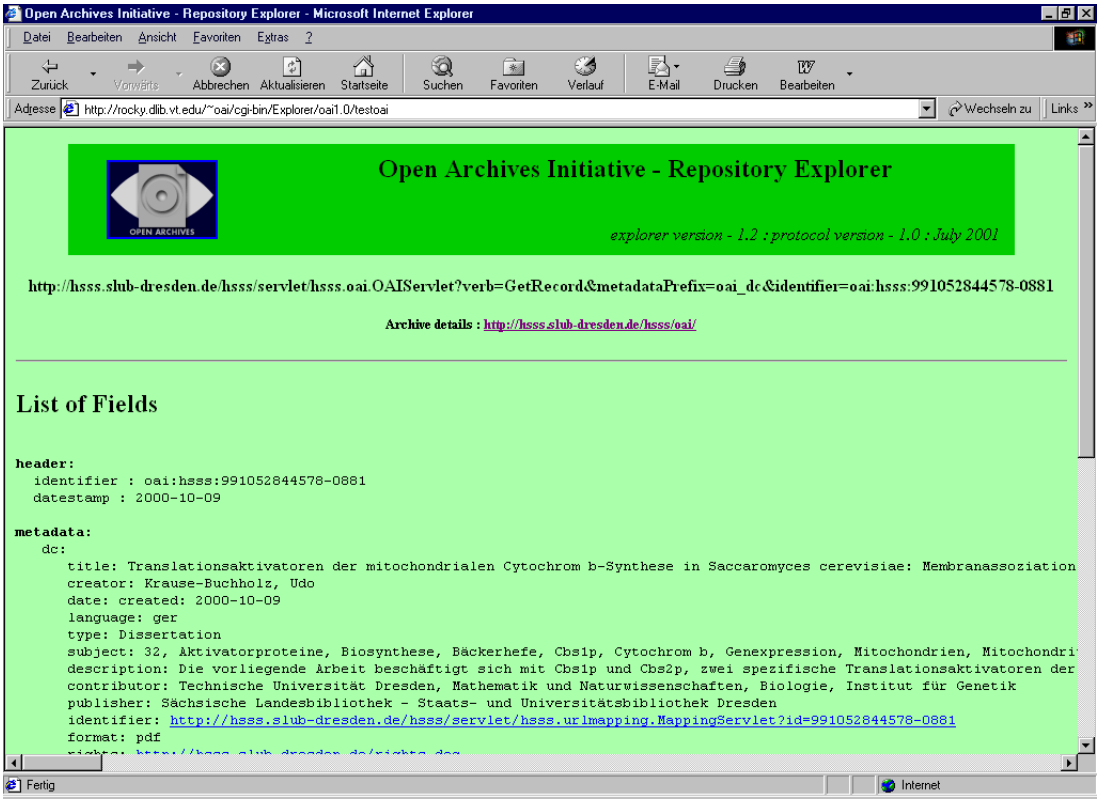
Open Archives Initiative - Repository Explorer - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://rocky.dlib.vt.edu/~oai/cgi-bin/Explorer/oai1.0/testoai
Wechseln zu Links
Open Archives Initiative - Repository Explorer
explorer version - 1.2 : protocol version - 1.0 : July 2001
http://hsss.slub-dresden.de/hsss/servlet/hsss.oai.OAIServlet?verb=GetRecord&metadataPrefix=oai_dc&identifier=oai:hsss:991052844578-0881
Archive details : http://hsss.slub-dresden.de/hsss/oai/
List of Fields
header:
identifier : oai:hsss:991052844578-0881
datestamp : 2000-10-09
metadata:
dc:
title: Translationsaktivatoren der mitochondrialen Cytochrom b-Synthese in Saccaromyces cerevisiae: Membranassoziation
creator: Krause-Buchholz, Udo
date: created: 2000-10-09
language: ger
type: Dissertation
subject: 32, Aktivatorproteine, Biosynthese, Bäckerhefe, Cbs1p, Cytochrom b, Genexpression, Mitochondrien, Mitochondri
description: Die vorliegende Arbeit beschäftigt sich mit Cbs1p und Cbs2p, zwei spezifische Translationsaktivatoren der
contributor: Technische Universität Dresden, Mathematik und Naturwissenschaften, Biologie, Institut für Genetik
publisher: Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden
identifier: http://hsss.slub-dresden.de/hsss/servlet/hsss.urlmapping.MappingServlet?id=991052844578-0881
format: pdf
Fertig
Internet

Abbildung 64 *- OAI Repository Explorer: Anzeige eines Records*

Die Abbildung zeigt die Repräsentation eines *Records*, als erstes werden die Informationen `identifier` und `datestamp` angezeigt, die aus dem Header der Protokollantwort stammen. Anschließend folgt ein Metadaten-Abschnitt, der den *Record* inhaltlich beschreibt. Hier wurde die XML-Struktur der Antwort bereits in eine etwas leserlichere HTML-Struktur umgesetzt. Das verwendete Metadatenformat ist *Dublin Core* und die einzelnen Kategorien sind von dem *Repository Explorer* bereits in der Form „DC-Kategorie:“ hevorgehoben, außerdem wurde der Verweis auf die absolute URL des *Records* bereits entsprechend verlinkt, so dass ein Klick zur entsprechenden Frontdoor führen würde.

Außerdem wurden von den Mitarbeitern des Referats 5.3 die notwendigen Informationsseiten für das OAI-*Repository* vorbereitet. Dazu zählen Seiten mit Informationen zu den rechtlichen Bedingungen im Umgang mit den Metadaten bzw. mit den eigentlichen Dokumenten und allgemeine Informationsseiten mit Inhaltsbeschreibungen.

Abbildung 65 zeigt die zentrale OAI-Informationsseite des HSSS, welche u.a. eine Inhaltsbeschreibung des Archivs in englisch und deutsch enthält:

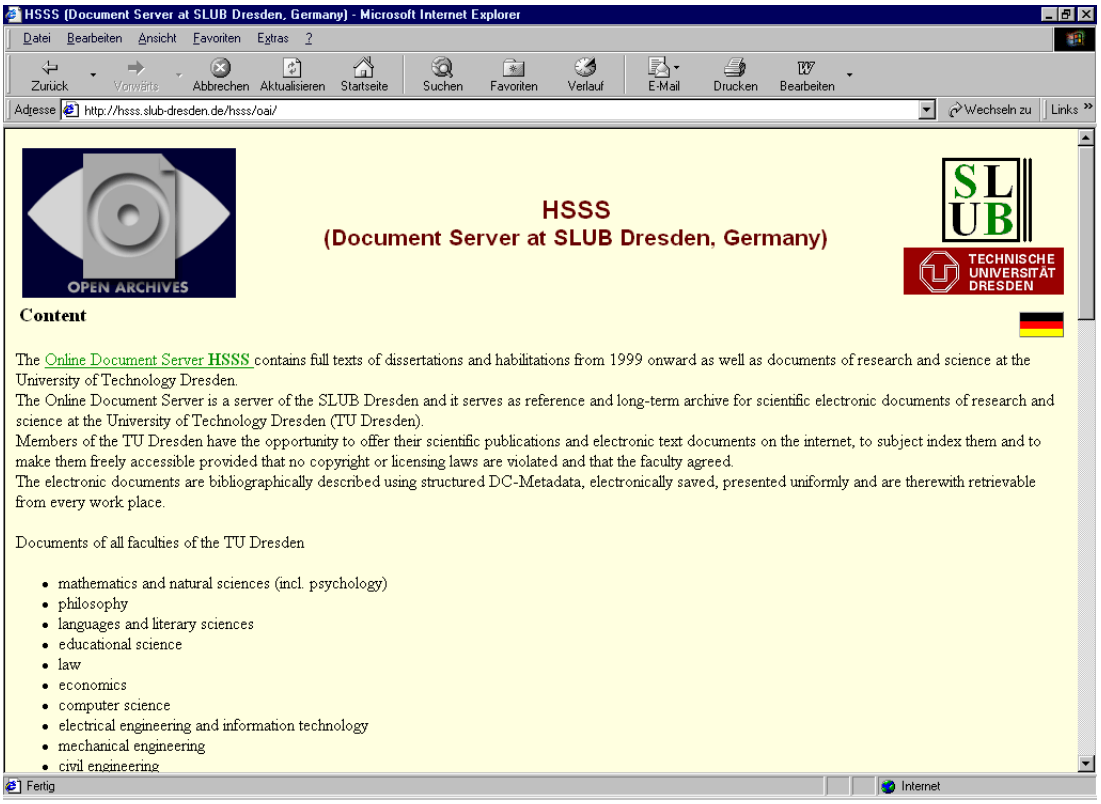
HSSS (Document Server at SLUB Dresden, Germany) - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen Favoriten Verlauf E-Mail Drucken Bearbeiten
Adresse http://hsss.slub-dresden.de/hsss/oai/
Wechseln zu Links
OPEN ARCHIVES
HSSS
(Document Server at SLUB Dresden, Germany)
SLUB
TECHNISCHE UNIVERSITÄT DRESDEN
Content
The Online Document Server HSSS contains full texts of dissertations and habilitations from 1999 onward as well as documents of research and science at the University of Technology Dresden.
The Online Document Server is a server of the SLUB Dresden and it serves as reference and long-term archive for scientific electronic documents of research and science at the University of Technology Dresden (TU Dresden).
Members of the TU Dresden have the opportunity to offer their scientific publications and electronic text documents on the internet, to subject index them and to make them freely accessible provided that no copyright or licensing laws are violated and that the faculty agreed.
The electronic documents are bibliographically described using structured DC-Metadata, electronically saved, presented uniformly and are therewith retrievable from every work place.
Documents of all faculties of the TU Dresden
mathematics and natural sciences (incl. psychology)
philosophy
languages and literary sciences
educational science
law
economics
computer science
electrical engineering and information technology
mechanical engineering
civil engineering
Fertig
Internet

Abbildung 65 *- OAI-Informationsseite des HSSS*

7.3. Einschätzung

Im Rahmen der Diplomarbeit konnte das OAI-Protokoll der Version 1.1 vollständig implementiert werden. Zusätzlich wurden sogar optionale Anforderungen wie etwa die Unterstützung von *Sets* und *Resumption Token* realisiert. Durch die XML-basierte Konfigurationsstruktur `oaisets.xml` bzw. die Konfigurationsdatei `web.xml` lassen sich zudem viele Parameter der Protokoll-Implementation von außen steuern. Da alle Anforderungen des Protokolls realisiert wurden und die Implementation zusätzlich fast alle der optionalen "Rubustness Tests"[L-OAd] meistern sollte, steht einer Registrierung des *Repository* bei der OAI als *Data Provider* nichts im Weg. Damit kommt der HSSS der wichtigen Anforderung nach, ein offenes, interoperables Archiv zu sein, welches in einem dezentralen Verbund übergreifend national und international genutzt werden kann.

8. Zusammenfassung und Ausblick

Inhalt des Kapitels:

8.1. Zusammenfassung 220
8.2. Verbesserungsmöglichkeiten 222
8.2.1. Verbesserungsmöglichkeiten des Gesamtsystems 222
8.2.2. Verbesserungsmöglichkeiten für den MetadatenGenerator . . 224
8.2.3. Verbesserungsmöglichkeiten für die AdminTools 225
8.2.4. Verbesserungsmöglichkeiten für die AccessTool / myHSSS . . 226
8.3. Ausblick 227

8.1. Zusammenfassung

Die im Rahmen des Diploms erreichten Ergebnisse können sich durchaus sehen lassen und sollen in diesem Abschnitt chronologisch erläutert und zusammengefasst werden. Dabei ergeben sich möglicherweise Ansatzpunkte für Verbesserungen und Weiterentwicklungen, die dann im folgenden Abschnitt wieder aufgegriffen werden.

Zunächst beschäftigte sich die Diplomarbeit relativ ausführlich mit dem bibliothekarischen Umfeld, insbesondere in den Bereichen digitale Archivierung, elektronisches Publizieren und Dokumentenmanagement. Dabei konnten wichtige Erkenntisse für die spätere Entwicklung des HSSS gewonnen werden. Leider musste hier aber auch die Feststellung getroffen werden, dass die bestehenden Informationen und Hilfsangebote nicht optimal sind und insbesondere keine übergeordnete Koordination von Ressourcen, Initiativen und Projekten erkennbar wird, so dass der Laie schnell die Übersicht verliert und auch generell gefragt werden muss, ob die gegenwärtige Situation der Entwicklung von modernen, interaktiven und interoperablen wissenschaftlichen Archiven zuträglich ist.

Mit diesen Erkenntnissen wurde begonnen, die Gesamtarchitektur eines modernen wissenschaflichen Dokumentenservers bzw. Hochschulschriftenservers zu entwickeln. Die Ansprüche dieser Architektur und insbesondere der aus ihr abgeleiteten Architekturvisionen gehen über die Möglichkeiten einer Diplomarbeit hinaus und sollten deshalb auch für zukünftige Erweiterungen und Verbesserungen eine gewisse Gültigkeit behalten. Parallel dazu wurde der Spezifikation der Arbeitsabläufe und Geschäftsmodelle, die sich für den Hochschulschriftenserver ergaben, besondere Aufmerksamkeit geschenkt. Die beiden entstandenen Geschäftsmodelle für die beiden derzeit vom HSSS unterstützten Dokumentenarten wurden mit allen beteiligten internen und externen Institutionen abgestimmt und konnten bereits ausführlich und erfolgreich getestet werden. Aufbauend auf diesen Ergebnissen erfolgte die Konkretisierung und Priorisierung der Architekturvisionen auf die durch die von der Diplomarbeit zu realisierenden Bereiche. So konnten für die Gesamtanwendung, aber auch für die drei Teilanwendungen konkrete Anforderungen aufgestellt werden.

Nun erfolgte die objektorientierte Analyse der Gesamtanwendung und die Überarbeitung der bereits in der Belegarbeit entstandenen Version des *MetadatenGenerators*. Dabei wurde auf eine möglichst lose Kopplung der einzelnen Komponenten sowie auf die Integration von passenden Entwurfsmustern geachtet. Für die gesamte Anwendung

und insbesondere für den *MetadatenGenerator* wurde dabei ein flexibles Konzept realisiert, das es ermöglicht, viele Informationen in externe, XML-basierte Konfigurationsstrukturen auszulagern, so dass wesentliche Teile der Anwendung konfigurierbar sind, ohne dass in den Quellcode des bestehenden Systems eingegriffen werden muss. Bei dem *MetadatenGenerator* sind zusätzlich viele Bereiche des Layouts sowie einige Bedingungen bei der Validierung der Metadaten einstellbar. Damit geht das Design dieser Teilanwendung über einen normalen „Straight Forward"-Ansatz weit hinaus. Auch die Implementation konnte mit Hilfe zweier Auszubildender zusätzliche Funktionalitäten realisieren, die im Abschnitt 4.4 erläutert wurden.

Anschließend wurde mit den *AdminTools* die umfangreichste der drei Teilanwendungen realisiert. Dabei konnten alle vorher spezifizierten Arbeitsabläufe von der Teilanwendung technisch unterstützt werden. Gleichzeitig erlauben die *AdminTools*, wichtige Teile des Systems und deren Verhalten durch die integrierten Änderungsmöglichkeiten von Konfigurationsstrukturen und XSL-Stylesheets zu beeinflussen. Um die vielfältigen Anforderungen an diese Teilanwendung befriedigen zu können, wurden eine Reihe von externen API's bzw. Technologien integriert (siehe dazu auch Abschnitt D im Anhang). Innerhalb dieses Entwicklungsprozesses wurde auch das eigentliche Datenbankschema der Gesamtanwendung spezifiziert und das persistente, datenbankbasierte Einstellen von Metadatensätzen realisiert.

Danach begann mit der Realisierung der URL-Abbildung die Entwicklung der dritten Teilanwendung. Die geforderten Funktionalitäten konnten dabei durch die Integration einer externen Volltextsuchmaschine relativ elegant realisiert werden. Auf der Clientseite entstand mit der Umsetzung der dritten Architekturvision ein interessantes, vom Nutzer skalierbares Design, welches sich durch ein zusätzlich implementiertes JSP-Komponentenkonzept sehr flexibel gestalten lässt.

Als letztes konnte mit der Implementation des OAI-Protokolls eine Anbindung des HSSS an andere OAI-Archive realisiert werden, so dass der Dokumentenserver national und international im Sinne eines OAI-*Data Provider* genutzt werden kann. Darüberhinaus entstanden in Zusammenarbeit mit den beiden Auszubildenden eine Reihe von Dokumentationen (u.a. zu den Konfigurationsstrukturen, zum *MetadatenGenerator*, zur Hilfe des *MetadatenGenerators* und zu System- und Installationsanforderungen des HSSS).

8.2. Verbesserungsmöglichkeiten

Dieser Abschnitt soll die vielen Hinweise zu Verbesserungsmöglichkeiten, die im Verlauf dieser Arbeit gemacht wurden, noch einmal systematisch zusammenfassen. Dabei werden die Verbesserungen und Erweiterungen in solche eingeteilt, die sich generisch auf das Gesamtsystem beziehen und in solche, die den einzelnen Teilanwendungen zuordenbar sind. Wenn eine Verbesserung respektive Erweiterung bereits explizit in einem vorangegangenen Abschnitt beschrieben wurde, wird sie in dieser Zusammenfassung nur noch einmal kurz genannt und an die entsprechende Stelle verwiesen.

8.2.1. Verbesserungsmöglichkeiten des Gesamtsystems

- Eine erste Form der Erweiterung betrifft die Grundlage des Gesamtsystems: die Basistechnologien, auf denen alle anderen Teile der Anwendung aufbauen, sollten stets so aktuell wie möglich gehalten werden. Zu diesen Basistechnologien zählen zum Beispiel die Programmiersprachenumgebung (also das JSDK), das Datenbankverwaltungssystem und der Servlet-Container. Die nächste Version des JSDK (Version 1.4) wird kurze Zeit nach Fertigstellung der Diplomarbeit erscheinen und bietet einige Neuerungen im XML-Bereich sowie eine bessere Performance und Stabilität der Virtuellen Maschine (VM) sowie Verbesserungen der Hot-Spot-Technologie. Außerdem ist die Java WebStart - Technologie, von der bereits in Abschnitt 3.3.3.2 die Rede war, in dieses Release integriert worden. Der Datenbankserver *Adaptive Server Enterprise* ist bereits kurz vor dem Ende der Diplomarbeit in der neuen Version 12.5 vorgestellt wurden, allerdings war die Zeit zu kurz, um die Anwendung noch umzustellen. Die neue Version des Datenbankservers bietet jedoch eine Reihe von Vorteilen, insbesondere was die Java- bzw. XML-Unterstützung betrifft. Der ServletContainer *Tomcat* macht mit der nächsten Versionsnummer 4.0 nicht nur einen großen Versionssprung, sondern implementiert auch gleich die jeweils aktuellsten Spezifikationen der Servlet- bzw. JSP-Technologie. Dadurch ergeben sich neue, interessante Funktionalitäten wie zum Beispiel ServletFilter. Aber auch die Umgebung des Servlet-Containers wird sich stark verändern, vor allem was deren Benutzerfreundlichkeit, die Dokumentation und die Integration von Werkzeugen (wie etwa JSPC) betrifft.
- Sicherheitsfunktionalitäten, wie etwa das Zertifizieren des Hochschulschriftenservers bzw. der Hochschulschriften selbst, sind schon in Abschnitt 2.4.5 als eine wichtige, in einer nächsten Ausbaustufe des HSSS zu realisierende Erweiterung in Zusammenhang mit der Unterstützung einer *Public Key Infrastructure* (PKI)

angesprochen worden. Deshalb soll diese Erweiterungsmöglichkeit an dieser Stelle nicht weiter vertieft werden.

- Auch die Unterstützung einer Auszeichnungssprache wie die in Abschnitt 2.4.3.1 vorgestellte DiML wurde bereits in Abschnitt 2.4.5 vorgeschlagen. Dies setzt aber eine organisatorische und technische Erweiterung der Arbeitsabläufe voraus sowie die Schulung von Mitarbeitern und Universitätsangehörigen, außerdem wäre eine enge Kooperation mit den einzelnen Fakultäten notwendig, um die entsprechende Formatvorlagen etablieren zu können.

- In einer nächsten Ausbaustufe ist zu überdenken, ob das Gesamtsystem nicht auf einen J2EE-Application Server portiert werden sollte[DeP00]. Durch die Komplexität, die die Gesamtanwendung inzwischen erreicht hat, erscheint dieser Schritt sinnvoll.

- Die Gesamtanwendung des HSSS hat schon jetzt einen beträchtlichen Umfang, auch was die Zahl der Klassen und JSP-Dateien betrifft. Dadurch, dass bei dem ersten Zugriff nach dem Neustart des Servlet-Containers auf ein JSP dieses in einem zweistufigen Prozess erst kompiliert werden muss, dauert dieser erste Zugriff im Vergleich zu allen anderen relativ lange. Kann man bei wenigen JSP's diese für den Nutzer unangenehme Situation noch durch ein manuelles Ansteuern aller JSP's verhindern, ist das bei größeren Systemen wie dem HSSS (mit knapp 200 JSP's) schlicht unmöglich. Zu diesem Zweck dient das Werkzeug JSPC (JSP Compiler), welches alle JSP's noch vor dem Hochfahren des Servlet-Containers kompilieren kann, so dass diese mit der gewohnten Geschwindigkeit ausführbar sind. Allerdings ist dieses Werkzeug bis jetzt nur sehr dürftig dokumentiert und hat bei kurzen Tests noch einige Probleme bereitet. Trotzdem sollte in die Nutzung dieses Werkzeuges investiert werden, da es sich um so mehr auszahlt, je größer die Gesamtanwendung wird.

- In Abschnitt 3.3.3.1 wurden sogenannte TagLibs vorgestellt und die Vorteile aufgezählt, die der Einsatz dieser Technologie mit sich bringen würde. Zwar ist die Entwicklung von flexiblen, wiederverwendbaren TagLibs nicht trivial, trotzdem sollte sie in einer nächsten Ausbaustufe wenigstens getestet werden. Sinnvoll ist der Einsatz grundsätzlich bei jeder der drei Teilanwendungen.

- In einer nächsten Ausbaustufe sollten auf jeden Fall die guten Internationalisierungsmöglichkeiten von Java genutzt werden, um durch entsprechende `Locales` verschiedene Sprachen und Formate zu unterstützen. Diese Erweiterung muss mit den Konfigurationsstrukturen synchronisiert werden. Ziel sollte es sein, zumindest Englisch durchgehend als alternative Sprache (in den Benutzeroberflächen etc.) zu unterstützen. Besonders wichtig wäre diese Unterstützung für die Teilanwendungen *MetadatenGenerator* und *AccessTools / myHSSS*.

- Die in Abschnitt 2.4.5 vorgestellte Java WebStart-Technologie kann dazu führen, dass die Vorteile von Web-Anwendungen (leichtes Deployment, gute Wartbarkeit) und die von traditionellen, clientseitigen Anwendungen (reichhaltige Benutzerschnittstelle, vertraute Umgebung) zusammenkommen können. Allerdings hat die Realisierung einer WebStart-basierten Anwendung keine überragende Priorität, da bereits funktionstüchtige, webbasierte Teilanwendungen existieren und eine solche Realisierung niemals ersetzend, sondern alternativ angeboten werden sollte. Am sinnvollsten wäre eine solche Alternative für die zweite Teilanwendung *AdminTools*.

8.2.2. Verbesserungsmöglichkeiten für den MetadatenGenerator

Die Mehrheit der hier aufgeführten Verbesserungen bzw. Erweiterungen wurde bereits in Abschnitt 4.5 auf Seite 142 diskutiert, der Vollständigkeit halber sollen sie hier trotzdem noch einmal aufgeführt werden:

- Die Navigation durch die einzelnen Masken des *MetadatenGenerators* sollte weiter verbessert werden, indem eingegebene Daten auch bei einem Klick auf „Zurück“ übernommen, allerdings nicht validiert werden. Das hat den Vorteil, dass die Daten vom Nutzer in diesem Fall nicht umsonst eingegeben werden, verursacht aber einige Schwierigkeiten in der Behandlung auf der Serverseite.

- Eine weitere Verbesserung der Navigation wäre die Einführung einer kontextsensitiven SiteMap, die die augenblickliche Position in der Abfolge der Masken grafisch visualisiert und gleichzeitig selbst zur Navigation genutzt werden kann. Dies dürfte allerdings nur in Abhängigkeit der aktuellen Situation geschehen[1] und stellt insbesondere an den serverseitigen Teil des *MetadatenGenerators* hohe Anforderungen.

- Das Layout der Teilanwendung könnte weiter verbessert werden, indem die positiven Erfahrungen mit der serverseitigen Skalierung der Benutzeroberfläche, die

[1] So sollte der Nutzer nicht von Seite 1 auf die Übersicht springen dürfen, sondern müsste zunächst alle Masken in der vorgegebenen Reihenfolge durchlaufen, hat er diese aber quasi „freigeschaltet“, darf er sich frei bewegen und gegebenenfalls auch Masken überspringen. Dies käme insbesondere dem Anwendungsfall zugute, bei dem ein Nutzer ausgehend von der Übersichtsmaske eine Korrektur auf Seite 1 vornimmt und anschließend mit Hilfe der grafischen SiteMap direkt zur Übersichtsseite „zurückspringen“ kann. Bisher muss er alle dazwischenliegenden Masken sequentiell durchlaufen.

bei der Entwicklung der Teilanwendung *myHSSS* gewonnen werden konnten, auf den *MetadatenGenerator* übertragen werden.

- Die kontextsensitive Hilfe des *MetadatenGenerators* könnte dahingehend verbessert werden, dass die Beschreibungen und Beispiele zu den einzelnen Subelementen, die bisher als JSP-Komponenten realisiert sind, in die Datenbank ausgelagert werden. Gleichzeitig wäre die Entwicklung einer Anwendung sinnvoll, die es erlaubt, diese ausgelagerten Informationen zu bearbeiten.

8.2.3. Verbesserungsmöglichkeiten für die AdminTools

Die Mehrheit der hier aufgeführten Verbesserungen bzw. Erweiterungen wurde bereits in Abschnitt 5.4 auf Seite 174 diskutiert, der Vollständigkeit halber sollen sie hier trotzdem noch einmal aufgeführt werden:

- Die bereits vorhandenen administrativen Möglichkeiten der Teilanwendung *AdminTools* sollten erweitert und stabilisiert werden. Es könnten außerdem weitere Funktionalitäten integriert werden, die im Rahmen der Diplomarbeit noch nicht in Angriff genommen oder nur angedeutet werden konnten. Beispielsweise ist bereits die Oberfläche eines webbasierten, interaktiven SQL-Clients erstellt worden, es fehlt aber noch dessen Anbindung an die Datenbank. Weitere denkbare Funktionalitäten wären die Integration eines Upload-Mechanismus für die Hochschulschriften in das temporäre Dokumentenverzeichnis des HSSS und eine webbasierte E-Mail-Schnittstelle zu dem E-Mail-Account des HSSS (zur Zeit hsss@slub-dresden.de). Dadurch wären dann alle Arbeitsablauf-Schritte von einem beliebigen Punkt der Erde realisierbar, einen Browser und eine HTTP- respektive SSL-Verbindung vorausgesetzt.
- Die Übersichtsseiten, auf denen alle temporären bzw. alle eingestellten Metadatensätze präsentiert werden, sollten so erweitert werden, dass sie auch im Falle großer Metadatensatzzahlen eine übersichtliche Darstellung ermöglichen. Dazu ist es höchstwahrscheinlich notwendig, nur einen Teil der Metadatensätze anzuzeigen. Eine weitere Möglichkeit wäre, die Ordnung bzw. Reihenfolge der Metadatensätze dynamisch nach einer der vier angezeigten Informationen (Titel, Bearbeitungsnummer, Status und letztes Bearbeitungsdatum) neu zu erstellen.
- Die XML-basierten Strukturen, die zur Zeit im Rahmen der Administrationsintegration noch direkt geändert werden müssen, sollten durch eine zwischengeschaltete Anwendung vor den Benutzern verborgen werden. Diese Anwendung sollte

die Konfigurationsstrukturen zum einen in einer benutzerfreundlichen Art und Weise aufbereiten (durch grafische Formulare etc.), zum anderen sollte sie die Konfigurationsstrukturen vor direkten Manipulationen schützen, indem sie Änderungen zunächst auf deren Korrektheit überprüft. So kann verhindert werden, dass die Gesamtanwendung infolge korrupter Konfigurationsstrukturen nicht verfügbar ist.

8.2.4. Verbesserungsmöglichkeiten für die AccessTool / myHSSS

Die Mehrheit der hier aufgeführten Verbesserungen bzw. Erweiterungen wurde bereits in Abschnitt 6.4 auf Seite 205 diskutiert, der Vollständigkeit halber sollen sie hier trotzdem noch einmal aufgeführt werden:

- Die gegenwärtig benutzte explizite Abbildung von absoluten auf reale URL's ist zu überdenken und gegebenenfalls durch eine implizite Abbildung zu ersetzen. Dies sollte allerdings erst bei wesentlich größeren Datenmengen erforderlich sein, wobei die Frage ist, ob diese vom HSSS jemals erreicht werden[2]. Bei einer Umstellung ist darauf zu achten, dass alle bereits veröffentlichten absoluten URL's weiterhin unterstützt werden müssen.
- Die Volltextsuchmaschine *Lucene* ist noch besser an die deutsche Sprache anzupassen. Dazu sollte als erstes die innerhalb der Klasse `HsssStopAnalyzer` realisierte Stopwortliste auf Vollständigkeit überprüft werden. Zusätzlich wäre die Integration eines deutschen Stemming-Algorithmus für die Suche nützlich, dieser muss allerdings sehr wahrscheinlich selbst implementiert werden.
- Die bisher nur sehr primitiv realisierte Navigation in den Metadaten sollte deutlich erweitert werden. Ein möglicher Ansatz dazu wurde in Abschnitt 6.4 genannt.
- Die Benutzeroberfläche *myHSSS* sollte zu einem Portal ausgebaut werden, so dass sie der zentrale Interaktionspunkt des HSSS wird. Dazu ist die persistente, datenbankbasierte Registrierung in Verbindung mit Personalisierungsmöglichkeiten zu realisieren. Unter diesen Voraussetzungen ergeben sich eine Reihe von interessanten Dienstleistungen, die den potentiellen Nutzern angeboten werden können. Einige dieser Möglichkeiten wurden in Abschnitt 6.4 angedeutet.

[2]Der explizite Abbildungsansatz sollte für Datenmengen im Bereich von 1000 bis 10000 Hochschulschriften ausreichen.

8.3. Ausblick

Im Verlauf dieser Diplomarbeit konnte für die SLUB ein rundes System erstellt werden, das allen primären Anforderungen an einen Hochschulschriftenserver gerecht wird und darüber hinaus sehr interessante Ideen andeutet oder sogar umsetzt. Außerdem wurde es auf der Basis von modernen, zukunftsträchtigen Technologien und zudem unschlagbar preiswert entwickelt. Dadurch bietet das bestehende System ideale Voraussetzungen für die Entwicklung und das Betreiben einer Produktivversion des HSSS. Denn das System in seiner derzeitigen Form ist *kein* Produktivsystem im Sinne einer Anwendung, die einmal installiert für die nächsten Jahre ihren Dienst tut. Obwohl der Autor sich redliche Mühe gegeben hat, eine stabile Anwendung zu realisieren und durch regelmäßige Tests die Zahl der Fehler zu minimieren, ist sicher einsichtig, dass eine Anwendung dieser Komplexität, die in wenigen Monaten entstanden ist, nicht die Erfüllung der harten Anforderungen einer Produktivversion[3] garantieren kann.

Die Anwendung kann auch nicht alle Eventualitäten vorhersehen, die sich möglicherweise in den nächsten Monaten oder Jahren ergeben, obwohl sie aufgrund ihrer sehr guten Konfigurierbarkeit dafür besser gerüstet ist als die meisten anderen Anwendungen. Wenn die SLUB auch mittelfristig aus den entstandenen Entwicklungen Nutzen ziehen will, ist es absolut notwendig, dass die bestehende Anwendung kontinuierlich gewartet und weiterentwickelt wird[4]. Um eine solche Kontinuität zu gewährleisten, ist es nach Ansicht des Autors notwendig, für diese Aufgabe eine feste Stelle zu schaffen, die sich aufgrund des inhaltlichen Bezuges in organisatorischer Nähe des Referats 5.3 befinden sollte. Alles andere kommt einem langsamen, aber relativ sicheren Todesurteil für den HSSS gleich. Dann aber muss sich die SLUB fragen lassen, wie sie einen solche Dienst alternativ realisieren will. Noch einmal: alle Voraussetzungen dafür, dass die SLUB mit einem modernen Hochschulschriftenserver den ersten Schritt auf dem Weg zu einer zukunftsfähigen Hybridbibliothek[Wis01] mit einem Portfolio an digitalen Dienstleistungen unternehmen kann, sind durch die Diplomarbeit erfüllt. Ob diese Chance erfolgreich genutzt werden kann, hängt davon ab, wie konsequent die begonnene Entwicklung weitergeführt wird. Die Entscheidung darüber liegt selbstverständlich bei der SLUB.

[3] Das können die wenigsten Produktivversionen. Es war auch nicht die Aufgabe der Diplomarbeit, eine solche Version zu entwickeln.

[4] Dies gilt für jede Art von Software.

A. Metadatensätze des HSSS

Inhalt des Kapitels:

A.1. Beschreibungkriterien 230
A.2. Spezifikation der Metadaten 231
A.2.1. DC.TITLE 231
A.2.2. DC.CREATOR 232
A.2.3. DC.SUBJECT 233
A.2.4. DC.DESCRIPTION 235
A.2.5. DC.PUBLISHER 236
A.2.6. DC.CONTRIBUTOR 237
A.2.7. DC.DATE 238
A.2.8. DC.TYPE 239
A.2.9. DC.IDENTIFIER 239
A.2.10. DC.FORMAT 240
A.2.11. DC.SOURCE 241
A.2.12. DC.LANGUAGE 242
A.2.13. DC.RELATION 242
A.2.14. DC.RIGHTS 243
A.3. Spezifikation der temporären Metadaten 244

A.1. Beschreibungkriterien

Im folgenden Abschnitt werden die einzelnen *Subelements* der HSSS-Metadatensätze vorgestellt, getrennt nach DC-Kategorien. Zuerst sollen die Kriterien erläutert werden, anhand denen sie später detailliert beschrieben werden:

Kriterium	Kriterienbeschreibung
Bezeichnung	Kurze textuelle Beschreibung des Subelements
Beschreibung	längere Beschreibung der Funktion bzw. Aussage des Subelements
DC-Bezeichnung	die DC-Bezeichnung des Subelements nach dem DC-Standard
Parametername	der Name für dieses DC-Subelement, der als Zugriffs-Schlüssel auf den Parameter bei der Übertragung durch das HTTP-Protokoll verwendet wird
Metadatentyp	mögliche Typen sind : *"primary"*,*"secondary"* oder *"system"*. Vom Typ *primary* sind alle Metadaten, die von Autoren der Hochschulschrift zu erbringen sind. Vom Typ *secondary* sind Metadaten, die von Fachkräften der Bibliothek erstellt werden (Klassifikationen etc.). Vom Typ *system* schließlich sind Metadaten, die automatisch von der Anwendung erstellt bzw. vergeben werden.
Regulation	beschreibt, ob das Subelement obligatorisch (*"mandatory"*) oder optional (*"optional"*) ist
Häufigkeit	beschreibt, wie oft das entsprechende Subelement auftreten darf (min, max)
Längenrestriktion	beschreibt sinnvolle Längenangaben für die Subelements (min, max)

Tabelle A1 - Kriterien für die Metadatenbeschreibung

Die Metadatensätze für die beiden Dokumentenarten *Dissertation* und *Bericht* werden anhand der eben eingeführten Kriterien gemeinsam beschrieben. Sollten bei einzelnen Subelements bei einigen Kriterien Unterschiede bestehen, werden diese im Hinblick auf die entsprechende Dokumentenart ([**Dissertation**], [**Bericht**]) ausgewiesen.

A.2. Spezifikation der Metadaten

A.2.1. DC.TITLE

Bezeichnung	Originaltitel der Arbeit
Beschreibung	Der originale Titel der Dissertation / Habilitation. Unter Umständen auch ein fremdsprachlicher Titel (in Abhängigkeit von DC.Language).
DC-Bezeichnung	DC.Title
Parametername	dc_title
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A2 - HSSS-Metadatensatz : Originaltitel des Dokuments

Bezeichnung	Untertitel / weiterer Titel
Beschreibung	Eine Liste von weiteren Titeln, z. Bsp. Untertiteln.
DC-Bezeichnung	DC.Title.Alternative
Parametername	dc_title_alternative
Metadatentyp	primary
Regulation	optional
Anzahl	(0, 10)
Längenrestriktion	(10, 255)

Tabelle A3 - HSSS-Metadatensatz : Untertitel / weitere Titel

Bezeichnung	Übersetzter Titel
Beschreibung	Eine Liste von übersetzten Titeln.
DC-Bezeichnung	DC.Title.Translated
Parametername	dc_title_translated
Metadatentyp	primary
Regulation	optional
Häufigkeit	(0, 10)
Längenrestriktion	(10, 255)

Tabelle A4 - HSSS-Metadatensatz : Übersetzter Titel

A.2.2. DC.CREATOR

Bezeichnung	Name des Verfassers
Beschreibung	Der Name eines Verfassers.
DC-Bezeichnung	DC.Creator.PersonalName
Parametername	dc_creator_personalname
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(8, 60)

Tabelle A5 - HSSS-Metadatensatz : Name des Verfassers

Bezeichnung	Geburtsdatum des Verfassers
Beschreibung	Das Geburtsdatum eines Verfassers.
DC-Bezeichnung	DC.Creator.PersonalName.DateOfBirth
Parametername	dc_creator_personalname_dateofbirth
Metadatentyp	primary (nicht genutzt)
Regulation	optional
Häufigkeit	(0, 10)
Längenrestriktion	(10, 50)

Tabelle A6 - HSSS-Metadatensatz : Geburtsdatum des Verfassers

Bezeichnung	Geburtsort des Verfassers
Beschreibung	Der Geburtsort eines Verfassers.
DC-Bezeichnung	DC.Creator.PersonalName.PlaceOfBirth
Parametername	dc_creator_personalname_placeofbirth
textbfMetadatentyp	primary (nicht genutzt)
Regulation	optional
Häufigkeit	(0, 10)
Längenrestriktion	(4, 255)

Tabelle A7 - HSSS-Metadatensatz : Geburtsort des Verfassers

Bezeichnung	Adresse des Verfassers
Beschreibung	Die Adresse eines Verfassers.
DC-Bezeichnung	DC.Creator.PersonalName.Address
Parametername	dc_creator_personalname_address
Metadatentyp	primary (nicht genutzt)
Regulation	optional
Häufigkeit	(0, 10)
Längenrestriktion	(10, 255)

Tabelle A9 - HSSS-Metadatensatz : Adresse des Verfassers

A.2.3. DC.SUBJECT

Bezeichnung	Eigene Schlagworte (deutsch)
Beschreibung	Freie (selbstdefinierte) Schlagworte in deutscher Sprache.
DC-Bezeichnung	DC.Subject
Parametername	dc_subject_ger
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(3, 100)

Tabelle A10 - HSSS-Metadatensatz : Eigene Schlagworte (deutsch)

Bezeichnung	Eigene Schlagworte (englisch)
Beschreibung	Freie (selbstdefinierte) Schlagworte in englischer Sprache.
DC-Bezeichnung	DC.Subject
Parametername	dc_subject_eng
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(3, 100)

Tabelle A11 - HSSS-Metadatensatz : Eigene Schlagworte (englisch)

Bezeichnung	DNB-Klassifikation
Beschreibung	Klassifikation der DNB-Sachgruppe, zwingend erforderlich, weil für den Austausch benötigt.
DC-Bezeichnung	DC.Subject
Parametername	dc_subject_dnb
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(3, 100)

Tabelle A12 - HSSS-Metadatensatz : DNB-Klassifikation

Bezeichnung	RVK-Klassifikation
Beschreibung	Klassifikation der RVK.
DC-Bezeichnung	DC.Subject
Parametername	dc_subject_rvk
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(3, 100)

Tabelle A14 - HSSS-Metadatensatz : RVK-Klassifikation

Bezeichnung	SWD-Schlagworte
Beschreibung	Schlagworte nach der Schlagwortnormdatei (SWD).
DC-Bezeichnung	DC.Subject
Parametername	dc_subject_rvk
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 10)
Längenrestriktion	(3, 100)

Tabelle A15 - HSSS-Metadatensatz : SWD-Schlagworte

A.2.4. DC.DESCRIPTION

Bezeichnung	Beschreibung / Abstract (deutsch)
Beschreibung	Textuelle Inhaltsbeschreibung oder Abstract des Dokumentes in deutsch.
DC-Bezeichnung	DC.Description
Parametername	dc_description_ger
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(100, 2000)

Tabelle A16 - HSSS-Metadatensatz : Beschreibung (deutsch)

Bezeichnung	Beschreibung / Abstract (englisch)
Beschreibung	Textuelle Inhaltsbeschreibung oder Abstract des Dokumentes in englisch.
DC-Bezeichnung	DC.Description
Parametername	dc_description_eng
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(100, 2000)

Tabelle A17 - HSSS-Metadatensatz : Beschreibung (englisch)

A.2.5. DC.PUBLISHER

Bezeichnung	Name der Universitätsbibliothek
Beschreibung	Name der Institution, die für die Verbreitung bzw. die Bereitstellung der Hochschulschrift verantwortlich ist.
DC-Bezeichnung	DC.Publisher.CorporateName
Parametername	dc_publisher_corporatename
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A18 - HSSS-Metadatensatz : Name der Universitätsbibliothek

Bezeichnung	Adresse der Universitätsbibliothek
Beschreibung	Adresse der Institution, die für die Verbreitung bzw. die Bereitstellung der Hochschulschrift verantwortlich ist.
DC-Bezeichnung	DC.Publisher.CorporateName.Address
Parametername	dc_publisher_corporatename_address
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A19 - HSSS-Metadatensatz : Adresse der Universitätsbibliothek

A.2.6. DC.CONTRIBUTOR

Bezeichnung	Name der universitären Einrichtung
Beschreibung	Name der Institution, die für die Erstellung bzw. die Bewertung der Hochschulschrift verantwortlich ist (Universität, Fakultät, Fachbereich, Institut).
DC-Bezeichnung	DC.Contributor.CorporateName
Parametername	dc_contributor_corporatename
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(20, 255)

Tabelle A20 - HSSS-Metadatensatz : Name der universitären Einrichtung

Bezeichnung	Adresse der universitären Einrichtung
Beschreibung	Adresse der Institution, die für die Erstellung bzw. die Bewertung der Hochschulschrift verantwortlich ist (der Ort ist ausreichend).
DC-Bezeichnung	DC.Contributor.CorporateName.Address
Parametername	dc_contributor_corporatename_address
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A22 - HSSS-Metadatensatz : Adresse der universitären Einrichtung

Bezeichnung	Gutachter
Beschreibung	Die Gutachter der Hochschulschrift.
DC-Bezeichnung	DC.Contributor.Referee
Parametername	dc_contributor_referee
Metadatentyp	primary
Regulation	mandatory [**Dissertation**], nicht vorhanden [**Bericht**]
Häufigkeit	(1, 10)
Längenrestriktion	(5, 60)

Tabelle A23 - HSSS-Metadatensatz : Gutachter

Bezeichnung	Betreuer
Beschreibung	Der (die) Betreuer der Hochschulschrift.
DC-Bezeichnung	DC.Contributor.Advisor
Parametername	dc_contributor_advisor
Metadatentyp	primary
Regulation	mandatory [**Dissertation**], nicht vorhanden [**Bericht**]
Häufigkeit	(1,1)
Längenrestriktion	(5, 60)

Tabelle A24 - HSSS-Metadatensatz : Betreuer

A.2.7. DC.DATE

Bezeichnung	Tag der mündlichen Prüfung
Beschreibung	Datum der Promotion i.S. der Promotionsordnung (i.d.R ist das der Tag der mündlichen Prüfung).
DC-Bezeichnung	DC.Date.Accepted
Parametername	dc_date_accepted
Metadatentyp	primary
Regulation	mandatory [**Dissertation**], nicht vorhanden [**Bericht**]
Häufigkeit	(1, 1)
Längenrestriktion	(10, 50)

Tabelle A26 - HSSS-Metadatensatz : Tag der mündlichen Prüfung

Bezeichnung	Tag der Abgabe der Hochschulschrift (Begutachtung)
Beschreibung	Datum der Antragsstellung zur Promotion (i.d.R ist das der Tag der Abgabe der Dissertation).
DC-Bezeichnung	DC.Date.Submitted
Parametername	dc_date_submitted
Metadatentyp	primary
Regulation	mandatory [**Dissertation**], nicht vorhanden [**Bericht**]
Häufigkeit	(1, 1)
Längenrestriktion	(10, 50)

Tabelle A27 - HSSS-Metadatensatz : Tag der Abgabe

Bezeichnung	Tag der Abgabe der Hochschulschrift an die SLUB
Beschreibung	Datum der Abgabe der von der Universitätsbibliothek akzeptierten Pflichtexemplare.
DC-Bezeichnung	DC.Date.Created
Parametername	dc_date_created
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 50)

Tabelle A28 - HSSS-Metadatensatz : Tag der Abgabe (SLUB)

A.2.8. DC.TYPE

Bezeichnung	Dokumenten-Typ
Beschreibung	Typ des Dokumentes, zum Beispiel "Habilitation".
DC-Bezeichnung	DC.Type
Parametername	dc_type
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A29 - HSSS-Metadatensatz : Dokumenten-Typ

A.2.9. DC.IDENTIFIER

Bezeichnung	stabile Dokument-URL
Beschreibung	URL (stabil), über die das eingestellte Dokument erreichbar ist.
DC-Bezeichnung	DC.Identifier.Virtual
Parametername	dc_identifier_virtual
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A30 - HSSS-Metadatensatz : stabile Dokument-URL

Bezeichnung	reale Dokument-URL
Beschreibung	URL (real), über die das eingestellte Dokument erreichbar ist.
DC-Bezeichnung	DC.Identifier.Real
Parametername	dc_identifier_real
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A31 - HSSS-Metadatensatz : reale Dokument-URL

A.2.10. DC.FORMAT

Bezeichnung	HSSS-Format
Beschreibung	Format(e), in dem (denen) die Dokumente vorgehalten werden.
DC-Bezeichnung	DC.Format.Available
Parametername	dc_format_available
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, n)
Längenrestriktion	(3, 100)

Tabelle A32 - HSSS-Metadatensatz : HSSS-Format

Bezeichnung	Abgabe-Formate
Beschreibung	Format(e) des an den HSSS übermittelten Dokumentes (PDF, PS oder DVI).
DC-Bezeichnung	DC.Format.Submitted
Parametername	dc_format_submitted
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 3)
Längenrestriktion	(3, 100)

Tabelle A33 - HSSS-Metadatensatz : Abgabe-Formate

Bezeichnung	Original-Formate
Beschreibung	originale(s) Format(e) des an den HSSS übermittelten Dokumentes.
DC-Bezeichnung	DC.Format.Original
Parametername	dc_format_original
Metadatentyp	secondary
Regulation	optional
Häufigkeit	(0, n)
Längenrestriktion	(3, 100)

Tabelle A34 - HSSS-Metadatensatz : Original-Formate

A.2.11. DC.SOURCE

Bezeichnung	Veröffentlichungs-Titel
Beschreibung	Titel einer Veröffentlichung der Hochschulschrift.
DC-Bezeichnung	DC.Source.Title
Parametername	dc_source_title
Metadatentyp	primary
Regulation	optional
Häufigkeit	(0, 1)
Längenrestriktion	(10, 255)

Tabelle A35 - HSSS-Metadatensatz : Veröffentlichungs-Titel

Bezeichnung	Veröffentlichungs-ISBN
Beschreibung	ISBN (falls vorhanden) einer Veröffentlichung der Hochschulschrift.
DC-Bezeichnung	DC.Source.ISBN
Parametername	dc_source_isbn
Metadatentyp	primary
Regulation	optional
Häufigkeit	(0, 1)
Längenrestriktion	(10, 255)

Tabelle A36 - HSSS-Metadatensatz : Veröffentlichungs-ISBN

A.2.12. DC.LANGUAGE

Bezeichnung	Sprache des Dokuments
Beschreibung	Die Sprache(n), in der(denen) die Hochschulschrift verfasst ist.
DC-Bezeichnung	DC.Language
Parametername	dc_language
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(2, 50)

Tabelle A37 - HSSS-Metadatensatz : Sprache des Dokuments

A.2.13. DC.RELATION

Bezeichnung	Literaturquellen (extern)
Beschreibung	Verweise auf Literaturquellen (allgem. Dokumente, die mit der Hochschulschrift in Beziehung stehen).
DC-Bezeichnung	DC.Relation.Extern
Parametername	dc_relation_extern
Metadatentyp	primary
Regulation	optional
Häufigkeit	(0, n)
Längenrestriktion	(10, 255)

Tabelle A38 - HSSS-Metadatensatz : Literaturquellen (extern)

Bezeichnung	Literaturquellen (intern)
Beschreibung	URL einer internen Literaturquelle (i.d.R. Dokument auf HSSS).
DC-Bezeichnung	DC.Relation.Intern
Parametername	dc_relation_intern
Metadatentyp	secondary
Regulation	optional
Häufigkeit	(0, n)
Längenrestriktion	(10, 255)

Tabelle A39 - HSSS-Metadatensatz : Literaturquellen (intern)

A.2.14. DC.RIGHTS

Bezeichnung	RECHTLICHE BEDINGUNGEN
Beschreibung	Name, Version und URL der Datei, die die rechtlichen Bedingungen enthält.
DC-Bezeichnung	DC.Rights
Parametername	dc_rights
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 255)

Tabelle A40 - HSSS-Metadatensatz : rechtliche Bedingungen

A.3. Spezifikation der temporären Metadaten

Bezeichnung	Art der Dokumentübermittlung
Beschreibung	Art der Dokumentübermittlung, z. Bsp. per Mail, als CD oder durch Angabe einer URL.
DC-Bezeichnung	keine
Parametername	temp_transmission
Metadatentyp	primary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(2, 100)

Tabelle A41 - HSSS-Metadatensatz : Dokumentübermittlung

Bezeichnung	Eingangsdatum
Beschreibung	Datum, wann der Metadatensatz bei der SLUB eingegangen ist.
DC-Bezeichnung	keine
Parametername	temp_incoming
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 50)

Tabelle A42 - HSSS-Metadatensatz : Eingangsdatum

Bezeichnung	Abgabedatum des Dokuments
Beschreibung	Datum, wann das Dokument vom Autor abgegeben worden ist.
DC-Bezeichnung	keine
Parametername	temp_delivery
Metadatentyp	primary
Regulation	optional
Häufigkeit	(0, 1)
Längenrestriktion	(10, 50)

Tabelle A43 - HSSS-Metadatensatz : Abgabedatum des Dokuments

Bezeichnung	LETZTE BEARBEITUNG
Beschreibung	Datum, wann der Metadatensatz zuletzt von einem Mitarbeiter der SLUB bearbeitet worden ist.
DC-Bezeichnung	keine
Parametername	temp_lastprocessed
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 50)

Tabelle A44 - HSSS-Metadatensatz : Datum der letzten Bearbeitung

Bezeichnung	BEARBEITUNGSNUMMER / METADATENSATZ-ID
Beschreibung	Eindeutige Bearbeitungsnummer des Metadatensatzes.
DC-Bezeichnung	keine
Parametername	temp_processingid
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(10, 100)

Tabelle A45 - HSSS-Metadatensatz : Bearbeitungsnummer

Bezeichnung	GESAMT-BEARBEITUNGSSTATUS
Beschreibung	Gesamt-Bearbeitungsstatus des Metadatensatzes (auch nach außen sichtbar).
DC-Bezeichnung	keine
Parametername	temp_processingstate
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A46 - HSSS-Metadatensatz : Gesamt-Bearbeitungsstatus

Bezeichnung	Dokumenten-Bearbeitungsstatus
Beschreibung	Bearbeitungsstatus des Dokuments.
DC-Bezeichnung	keine
Parametername	temp_documentstate
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A47 - HSSS-Metadatensatz : Dokument-Bearbeitungsstatus

Bezeichnung	Metadaten-Bearbeitungsstatus
Beschreibung	Bearbeitungsstatus der Metadaten des Metadatensatzes.
DC-Bezeichnung	keine
Parametername	temp_metadatastate
Metadatentyp	secondary
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A48 - HSSS-Metadatensatz : Metadaten-Bearbeitungsstatus

Bezeichnung	Exportinformation DDB
Beschreibung	Informationen zum Metadatenexport an die DDB.
DC-Bezeichnung	keine
Parametername	temp_exportinformation_ddb
Metadatentyp	system
Regulation	optional
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A49 - HSSS-Metadatensatz : Exportinformation DDB

Bezeichnung	Exportinformation SWB
Beschreibung	Informationen zum Metadatenexport an die SWB.
DC-Bezeichnung	keine
Parametername	temp_exportinformation_swb
Metadatentyp	system
Regulation	optional
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A50 - HSSS-Metadatensatz : Exportinformation SWB

Bezeichnung	Dokumentinformationen
Beschreibung	Informationen zum Dokument (z. Bsp. relativer temporärer Pfad).
DC-Bezeichnung	keine
Parametername	temp_documentinformation
Metadatentyp	system
Regulation	mandatory
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A51 - HSSS-Metadatensatz : Dokumentinformationen

Bezeichnung	E-Mail Adresse des Autors
Beschreibung	E-Mail Adresse des Autors.
DC-Bezeichnung	keine
Parametername	temp_author_email
Metadatentyp	primary
Regulation	optional
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A52 - HSSS-Metadatensatz : E-Mail Adresse des Autors

Bezeichnung	Telefonnummer des Autors
Beschreibung	Telefonnummer des Autors.
DC-Bezeichnung	keine
Parametername	temp_author_telefon
Metadatentyp	primary
Regulation	optional
Häufigkeit	(1, 1)
Längenrestriktion	(5, 100)

Tabelle A53 - HSSS-Metadatensatz : Telefonnummer des Autors

Bezeichnung	Nachricht des Autors
Beschreibung	Mitteilung des Autors an die SLUB-Mitarbeiter.
DC-Bezeichnung	keine
Parametername	temp_author_message
Metadatentyp	primary
Regulation	optional
Häufigkeit	(1, 1)
Längenrestriktion	(5, 2000)

Tabelle A54 - HSSS-Metadatensatz : Nachricht des Autors

B. Konfigurationsstrukturen des HSSS

Inhalt des Kapitels:

B.1. Überblick 250
B.2. web.xml 251
B.3. hsssdata.xml 253
B.4. MetadataSetDissertation.xml / MetadataSetBericht.xml . 255
B.5. oaisets.xml 257

B.1. Überblick

Dieses Kapitel erläutert die Konfigurationsstrukturen des HSSS, welche sich aus den Anforderungen bzw. aus den Architekturvisionen ergeben haben und in relativ komplexer Form vorliegen. Dabei teilen sich die Konfigurationsstrukturen semantisch in verschiedene Bereiche auf, die in den folgenden Abschnitten getrennt behandelt werden. Zuerst wird die Konfigurationsdatei `web.xml` erläutert, welche die grundlegendsten Einstellungen enthält, die die Servlets der Web-Anwendung betreffen. Anschließend folgt die Konfigurationsstruktur `hsssdata.xml`, die insbesondere Daten beinhaltet, die die gesamte HSSS-Anwendung betreffen. Die Konfigurationsstrukturen `MetadataSetDissertation` bzw. `MetadataSetBericht` beschreiben die Metadatensätze der jeweiligen Dokumentenart. Die letzte Konfigurationsstruktur `oaisets.xml` spezifiziert die optionale *Set*-Struktur der OAI-Protokoll-Implementatiojn des HSSS.

B.2. web.xml

Bei der Konfigurationsdatei „`web.xml`“ handelt es sich um die von der Servlet-Spezifikation [DDC99] definierte Konfigurationsdatei für eine Web-Anwendung. Sie dient dazu, den Servlets einer solchen Anwendung bestimmte Informationen zukommen lassen zu können und wird von jedem Servlet-Container unterstützt, indem die Datei bei der Initialisierung der Web-Anwendung eingelesen und die enthaltene Information in bestimmten, den Servlets verfügbaren Objekten abgespeichert wird. Die formale Struktur dieser Konfigurationsdatei wurde bereits im Abschnitt 3.3.1.2 auf Seite 94 erläutert, weshalb an dieser Stelle auf den konkreten Inhalt der „`web.xml`“-Konfigurationsdatei des HSSS eingegangen werden soll, der Informationen für jedes Servlet der Gesamtanwendung enthält. Folgende Servlets existieren innerhalb der HSSS-Gesamtanwendung:

- `hsss.metadatagenerator.MetadataGeneratorServlet`
  Dieses Servlet fungiert als Controller der Teilanwendung *MetadatenGenerator*
- `hsss.admintools.AdminToolsServlet`
  Dieses Servlet fungiert als Controller der Teilanwendung *AdminTools*
- `hsss.accesstools.AccessToolsServlet`
  Dieses Servlet fungiert als Controller der Teilanwendung *AccessTools*
- `hsss.urlmapping.UrlMappingServlet`
  Dieses Servlet realisiert die Abbildung von stabilen (virtuellen) URL's auf reale URL's
- `hsss.oai.OAIServlet`
  Dieses Servlet implementiert das OAI-Protokoll, so dass der HSSS in einem Archivverbund als *Data Provider* fungieren kann

Für diese unterschiedlichen Servlets der Gesamtanwendung sind natürlich auch unterschiedliche Informationen innerhalb der entsprechenden Bereiche der Konfigurationsdatei „`web.xml`“ notwendig, auf die nicht in allen Einzelheiten eingegangen werden kann (detaillierte Informationen zu einzelnen Konfigurationsparametern finden sich aber in [JMS01]). Stattdessen sollen in der folgenden Aufzählung die wichtigsten Informationen genannt werden, die den Servlets (vor allem den Servlets der Teilanwendungen) auf diese Weise zur Verfügung stehen und somit flexibel konfigurierbar sind:

- Informationen, die das Servlet bzw. die Teilanwendung beschreiben (Name, Version, Release-Datum usw.)
- relative und absolute Pfade im Kontext der entsprechenden Teilanwendung
- Art der Persistenz (Datei-basiert, Datenbank-basiert)
- Angaben zum Logging (Bezeichnung der Log-Datei(en), Logging anschalten / ausschalten etc.)
- die Lokation weiterer Konfigurationsstrukturen (in Abhängigkeit von der Persistenzart entweder Dateipfade oder Tabellen-/Spaltenbezeichnungen)
- die Lokation von Stylesheets (Formatvorlagen, Transformationsbeschreibungen etc.)
- die Lokation der Daten (in Abhängigkeit von der Persistenzart entweder Dateipfade oder Tabellen-/Spaltenbezeichnungen)
- Informationen über das Layout der entsprechenden Teilanwendung

B.3. hsssdata.xml

Diese Konfigurationsstruktur kann in Abhängigkeit von der gewählten Persistenzart sowohl in Dateiform („`hsssdata.xml`“) als auch innerhalb einer entsprechenden Datenbanktabelle (in der Spalte „`hsssdataxml`“ der Tabelle `config_hsss`[1]) vorkommen und ist immer in Verbindung mit der zugehörigen Dokumentbeschreibungsdefinition (DTD) “`hsssdata.dtd`“ zu betrachten. Die Konfigurationsstruktur enthält Informationen, die sich auf Fähigkeiten / Funktionalitäten des HSSS als Ganzes beziehen. Die folgende Aufzählung nennt die wichtigsten Bereiche:

- `<HsssData:LANGUAGES> .... </HsssData:LANGUAGES>`
  Die vom HSSS unterstützten Sprachen nach ISO639-2 (Name, Acronym, Scheme)
- `<HsssData:FORMATS> .... </HsssData:FORMATS>`
  Die vom HSSS unterstützten Dokumenten-Formate (Name, Extension, Mime-Typ)
- `<HsssData:TYPES> .... </HsssData:TYPES>`
  Die vom HSSS unterstützten Dokumentenarten (`<HsssData.TYPE>`, z. Bsp Bericht) nebst zugehörigen Dokumententypen (`<HsssData.SUBTYPE>`, z. Bsp Forschungsbericht)
- `<HsssData:TRANSMISSION> .... </HsssData:TRANSMISSION>`
  Die vom HSSS unterstützten Übermittlungsarten von Dokumenten (z. Bsp. per URL, E-MAIL, CD usw.)
- `<HsssData:UNIVERSITY> .... </HsssData:UNIVERSITY>`
  Die vom HSSS unterstützte(n) Universität(en) inklusive der Fakultätsstruktur (`<HsssData:Faculty>`)
- `<HsssData:RIGHTS> .... </HsssData:RIGHTS>`
  Die rechtlichen Bedingungen des HSSS (Dokumentname, Version, URL)
- `<HsssData:PUBLISHERS> .... </HsssData:PUBLISHERS>`
  Informationen zur Universitätsbibliothek (SLUB)
- `<HsssData:EXPORT> .... </HsssData:EXPORT>`
  Informationen zum Metadatenexport (Name der Institutionen, SLUB-Informationen wie Identifikationsnummern bei der jeweiligen Institution, Zieladresse des Exports, Wartezeit auf Bestätigung usw.)

[1] siehe dazu auch Kapitel C im Anhang ab Seite 259

- `<HsssData:REALSTRUKTURES> .... </HsssData:REALSTRUKTURES>`
  Informationen zu Serverbezeichnungen und Verzeichnisstrukturen der Gesamtanwendung (Dokumentenverzeichnis, Spiegelverzeichnis, Temporäres Verzeichnis etc.)

- `<HsssData:STATES> .... </HsssData:STATES>`
  Die möglichen Zustände des HSSS (entscheidend für die *AdminTools*). Unterteilt werden Zustände in Dokumenten-Zustände (`type="document"`, Zustand der Dokumentenbearbeitung), Metadaten-Zustände (`type="metadata"`, Zustand der Metadatenbearbeitung) und Gesamtzustände (`type="overall"`), welche sich aus den ersten beiden Zuständen ableiten und auch nach außen sichtbar sind.

B.4. MetadataSetDissertation.xml / MetadataSetBericht.xml

Die Konfigurationsstrukturen `MetadataSetDissertation` bzw. `MetadatSetBericht` beschreiben die Metadatensätze der jeweiligen Dokumentenart und sind sich strukturell sehr ähnlich, weshalb sie an dieser Stelle auch gemeinsam behandelt werden können. Aufgrund der hierarchischen Struktur der *Dublin Core*-Metadaten und der Unterteilung in ähnlich strukturierte Kategorien bestehen auch innerhalb von `MetadataSetDissertation` bzw. `MetadatSetBericht` immer wiederkehrende Strukturen. Das auf der nächsten Seite abgebildete **Listing A1** zeigt eine solche Struktur exemplarisch.

Das Attribut `documenttype` des äußeren Tags beschreibt die Dokumentenart der Metadatensatzbeschreibung, möglich sind also „Dissertation“ und „Bericht“. Danach folgt eine Iteration von 15 `Element`-Tags[2], zu denen jeweils die Tags `ElementDCName` (Name der DC-Kategorie), `ElementDescription` (Beschreibung der DC-Kategorie), `ElementLabel` (Beschriftung der DC-Kategorie innerhalb des *MetadatenGenerators*) und `SubElements` gehören. Dieser letzte Tag enthält alle (selbstdefinierten) Subelements (`SubElement`) innerhalb der DC-Kategorien[3].

Zu jedem `SubElement` existieren wieder beschreibende Tags, so zum Beispiel `SubElementDCName` (DC-Name des Subelements), welches zusätzlich die Attribute `regulation` (obligatorisches oder optionales Subelement) und `metadatatyp` (Primär-Metadaten, Sekundär-Metadaten oder System-Metadaten) besitzt. `SubElementDescription` enthält die Beschreibung des Subelements, `SubElementLabel` die innerhalb des *Metadatengenerators* verwendete Beschriftung. `SubElementPaarameter` gibt die Bezeichnung des HTTP-Parameters an, der die Information zu diesem Subelement enthält. Das Tag `SubElementValidation` enthält mit den Tags `Length` und `Frequency` entsprechende Validierungsangaben.

[2] 14 der 15 *Dublin Core*-Kategorien sowie die temporären Metadaten, die als separates Element realisiert sind

[3] zur Zeit existiert also lediglich eine Hierarchietiefe von 2

```
<?xml version="1.0" encoding="ISO−8859−1" ?>
<MetadataSet documenttype="Dissertation">
    <Element>
        <ElementDCName>
            DC.Title
        </ElementDCName>
        <ElementDescription>
            Titel
        </ElementDescription>
        <ElementLabel>
            <![CDATA[
                Bitte geben Sie den Titel des Dokuments ein
            ]]>
        </ElementLabel>
        <SubElements>
            <SubElement>
                <SubElementDCName regulation="mandatory" metadatatype="primary">
                    DC.Title
                </SubElementDCName>
                <SubElementDescription>
                    Originaltitel
                </SubElementDescription>
                <SubElementLabel>
                    Originaltitel
                </SubElementLabel>
                <SubElementParameterName>
                    dc_title
                </SubElementParameterName>
                <SubElementValidation>
                    <Length min="15" max="2000">
                    </Length>
                    <Frequency min="1" max="1">
                    </Frequency>
                </SubElementValidation>
                <Parameter name="dc_title_language" type="DC.Language"></Parameter>
            </SubElement>
            <SubElement>
                ...
            </SubElement>
            ...
        </SubElements>
    </Element>
    ...
</MetadataSet>
```

***Listing A1** : formale XML-Struktur eines Metadatensatzes*

B.5. oaisets.xml

Die Konfigurationsstruktur `oaisets.xml` beschreibt die optionale *Set*-Struktur der Implementation des OAI-Protokolls. Diese *Set*-Structure entspricht den Vorschlägen der DINI für deutsche wissenschaftliche Archive[DINa01]. Die Umsetzung in eine XML-Struktur bot sich angesichts der starken XML-Ausrichtung der gesamten Anwendung an und hat den Vorteil, dass die Struktur von außen änderbar bleibt, ohne Eingriffe in die Anwendung vornehmen zu müssen.

Die Konfigurationsstruktur besteht aus hierarchisch geschachtelten `set`-Strukturen. Jede `set`-Struktur besteht dabei aus den Tags `setSpec` (Spezifikation des *Sets*) und `setName` (Name bzw. Beschreibung des *Sets*) sowie optional aus weiteren, rekursiven *Set*-Strukturen. Zur Zeit ist aber nur eine Hierarchietiefe von 2 realisiert. Die inhaltliche Bedeutung der einzelnen *Sets* wurde bereits in Kapitel 7.1 betrachtet.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<oaisets>
    <set>
        <setSpec>pubtype</setSpec>
        <setName>Publicationtype</setName>
        <set>
            <setSpec>Dissertation</setSpec>
            <setName>Dissertationen, Habilitationen</setName>
        </set>
        <set>
            <setSpec>Bericht</setSpec>
            <setName>Forschungsbereichte, Proceedings, etc.</setName>
        </set>
    </set>
    <set>
        <setSpec>dnb</setSpec>
        <setName>DNB Sachgruppe</setName>
        <set>
            <setSpec>1</setSpec>
            <setName>Wissenschaft und Kultur allgemein</setName>
        </set>
        <set>
            ...
        </set>
        ...
    </set>
</oaisets>
```

***Listing A2** : XML-Struktur der OAI-Sets*

C. Datenbank-Schema des HSSS

Inhalt des Kapitels:

C.1. Überblick 260
C.2. Schema 265

C.1. Überblick

Auf den folgenden Seiten soll das Datenbank-Schema des HSSS vollständig vorgestellt werden. Dazu werden als erstes alle beteiligten Tabellen mit ihrer jeweiligen Funktion aufgelistet (in den beiden folgenden Tabellen auf den nächsten Seiten). Danach erfolgt eine Übersicht über die erstellten gespeicherten Prozeduren innerhalb der Datenbank. Im anschließenden Abschnitt C.2 werden die Tabellen der HSSS-Datenbank jeweils einzeln aufgeführt und näher erläutert. Dabei werden die folgenden Attribute zur Beschreibung verwendet:

- **Name**
  Name der jeweiligen Spalte
- **Datentyp**
  Datentyp der jeweiligen Spalte
- **I**
  Steht für „Identify“. Identify sorgt für eine eindeutige Identifikationsnummer jedes Tupels in der Tabelle (bei anderen Datenbanken als „Auto Increment“ bezeichnet). Das Zeichen „$\surd$“ bedeutet JA, „-“ bedeutet NEIN.
- **P**
  Steht für „Primary Key“ (Primärschlüssel). Bedeutet, dass der Wert dieser Spalte das zugehörige Tupel innerhalb der Tabelle eindeutig beschreibt. Das Zeichen „$\surd$“ bedeutet JA, „-“ bedeutet NEIN.
- **M**
  Steht für „Mandatory“ (erforderlich). Bedeutet, dass ein Wert in dieser Spalte verlangt wird bzw. dass kein Null-Wert erlaubt ist. Das Zeichen „$\surd$“ bedeutet JA (NOT NULL), „-“ bedeutet NEIN (NULL).
- **Beschreibung**
  Beschreibt Besonderheiten (z. Bsp. Wertebereich) oder Funktion der jeweiligen Spalte, falls erforderlich.

Die Namen der Relationen wurden in der Regel aus der englischen Übersetzung des jeweiligen Gegenstandsbreiches und einem Prefix erstellt, dabei sind die in **Tabelle A55** gezeigten Prefix-Namen für Relationen verwendet worden.

Prefix	Beschreibung
pers	steht für „persistent“ und ist in allen Relationen enthalten, die zur persistenten Speicherung der Metadatensätze notwendig sind
temp	steht für „temporary“ und ist in allen Relationen enthalten, die temporäre Metadatensätze enthalten
stat	steht für „statistic“ und ist in allen Relationen enthalten, die statistische Informationen enthalten
config	steht für „configuration“ und ist in allen Relationen enthalten, die XML-basierte Konfigurationsstrukturen enthalten

Tabelle A55 *- Bedeutung der Relationen-Prefixe*

Tabelle	Beschreibung
config_hsss	Diese Tabelle enthält die Konfigurationsstruktur „hsssdata.xml“
config_metadatasets	Diese Tabelle enthält die XML-basierten Metadatensatzbeschreibungen für die vom HSSS unterstützten Dokumentenarten
stat_metadatasets	Diese Tabelle enthält statistische Informationen zu den einzelnen Metadatensätzen
temp_metadatasets	Diese Tabelle enthält die XML-basierten temporären Metadatensätze
temp_metadatasets_finished	Diese Tabelle enthält die XML-basierten Metadatensätze, die bereits eingestellt sind (als Proxy)
pers_address	Diese Tabelle enthält alle Adressen (z. Bsp. von Universitäten, Fakultäten etc.)
pers_date	Diese Tabelle enthält Datumsangaben
pers_description	Diese Tabelle enthält Beschreibungen (Abstracts) der Hochschulschriften
pers_format	Diese Tabelle enthält verwendete (Dokumenten-)Formate
pers_kindofdocument	Diese Tabelle enthält alle Dokumentenarten
pers_language	Diese Tabelle enthält Informationen zu verwendeten Sprachen
pers_person	Diese Tabelle enthält alle Personen
pers_publication	Diese Tabelle enthält Informationen zu Veröffentlichungen
pers_relation	Diese Tabelle enthält alle Relationen (zu anderen Ressourcen)
pers_rights	Diese Tabelle enthält alle verwendeten rechtlichen Bedingungen
pers_subject	Diese Tabelle enthält alle verwendeten Schlagworte bzw. Klassifikationen
pers_contributor	Diese Tabelle enthält alle beteiligten universitären Einrichtungen
pers_publisher	Diese Tabelle enthält alle beteiligten Universitätsbibliotheken

Tabelle A56 *- Tabellenübersicht (Teil 1)*

Tabelle	Beschreibung
pers_type	Diese Tabelle enthält alle Dokumententypen (die Dokumentenarten zugeordnet sind)
pers_metadatset	Diese Tabelle enthält eindeutige Attribute eines Metadatensatzes
pers_title	Diese Tabelle enthält alle verwendeten Titel
pers_fulltext	Diese Tabelle enthält die Volltexte zu den Hochschulschriften, falls verfügbar
pers_date_metadataset	Diese Tabelle enthält Zuordnungen von Daten zu Metadatensätzen und deren Typ (Prüfungstag, Abgabetag, Veröffentlichungsdatum)
pers_description_metadataset	Diese Tabelle enthält Zuordnungen von Beschreibungen zu Metadatensätzen und deren Typ (deutsche Beschreibung, englische Beschreibung)
pers_format_metadataset	Diese Tabelle enthält Zuordnungen von Dokumentenformaten zu Metadatensätzen und deren Typ (übermittelte Formate, Präsentationsformate, Originalformate)
pers_person_metadataset	Diese Tabelle enthält Zuordnungen von Personen zu Metadatensätzen und deren Typ bzw. Rolle (Autoren, Gutachter, Prüfer)
pers_publication_metadataset	Diese Tabelle enthält Zuordnungen von Veröffentlichungen zu Metadatensätzen
pers_relation_metadataset	Diese Tabelle enthält Zuordnungen von Relationen zu Metadatensätzen und deren Typ (interne Relationen, externe Relationen)
pers_subject_metadataset	Diese Tabelle enthält Zuordnungen von Schlagworten bzw. Klassifikationen zu Metadatensätzen und deren Typ (eigene Schlagworte(deutsch, englisch), Klassifikationen (DNB, RVK), Schlagworte nach SWD)
pers_title_metadataset	Diese Tabelle enthält Zuordnungen von Titeln zu Metadatensätzen und deren Typ (Originaltitel, alternative Titel, übersetze Titel)

Tabelle A57 *- Tabellenübersicht (Teil 2)*

gespeicherte Prozedur	Beschreibung
insert_type	wird verwendet beim Einstellen der Metadaten, liefert id des Dokumententyps und erstellt ihn gegebenenfalls
insert_language	wird verwendet beim Einstellen der Metadaten, liefert id der Sprache und erstellt sie gegebenenfalls
insert_publisher	wird verwendet beim Einstellen der Metadaten, liefert id des Veröffentlichers und erstellt ihn gegebenenfalls
insert_contributor	wird verwendet beim Einstellen der Metadaten, liefert id der universitären Einrichtung und erstellt sie gegebenenfalls
insert_rights	wird verwendet beim Einstellen der Metadaten, liefert id der rechtlichen Bedingungen und erstellt sie gegebenenfalls
insert_date	wird verwendet beim Einstellen der Metadaten, liefert id des Datums und erstellt es gegebenenfalls
insert_person	wird verwendet beim Einstellen der Metadaten, liefert id der Person und erstellt sie gegebenenfalls
insert_subject	wird verwendet beim Einstellen der Metadaten, liefert id des/-der Schlagworts/Klassifikation und erstellt es/sie gegebenenfalls
insert_title	wird verwendet beim Einstellen der Metadaten, liefert id des Titels und erstellt ihn
insert_format	wird verwendet beim Einstellen der Metadaten, liefert id des Formats und erstellt es gegebenenfalls
insert_relation	wird verwendet beim Einstellen der Metadaten, liefert id der Literaturquelle und erstellt sie gegebenenfalls
insert_publication	wird verwendet beim Einstellen der Metadaten, liefert id der Veröffentlichung und erstellt ihn gegebenenfalls
insert_metadataset	wird verwendet beim Einstellen der Metadaten, erstellt neuen Metadatensatz mit allen benötigten Fremdschlüsseln
delete_metadatset	löscht einen Metadatensatz mit allen zugehörigen Beschreibungen und Titeln
max_metadataset	liefert das größte id-Attribut der Tabelle pers_metadatasets
urlmapping	liefert alle Tupel (absolute URL, relative URL) ab einem bestimmten Wert für id
lucene_fields_metadataset	liefert alle id's und absoluten URL's für die Indexerstellung
lucene_fields_fulltext	liefert den Volltext für die Indexerstellung
lucene_fields_title	liefert alle Titel für die Indexerstellung
lucene_fields_subject	liefert alle Schlagworte / Klassifikationen für die Indexerstellung
lucene_fields_description	liefert alle Beschreibungen (Abstracts) für die Indexerstellung
lucene_fields_creator	liefert alle Autoren für die Indexerstellung
lucene_fields_advisor	liefert alle Betreuer für die Indexerstellung
lucene_fields_referee	liefert alle Gutachter für die Indexerstellung
lucene_fields_date	liefert das Veröffentlichungsdatum für die Indexerstellung
lucene_fields_type	liefert den Dokumententyp für die Indexerstellung

***Tabelle A58** - Übersicht der gespeicherten Prozeduren (Teil 1)*

gespeicherte Prozedur	BESCHREIBUNG
`processingstate_information`	liefert alle relevanten Informationen für die Bearbeitungsstatus-Anwendung des *MetadatenGenerators*
`oai_title`	liefert den Originaltitel für einen gegebenen OAI-Identifier
`oai_title_more`	liefert alternative und übersetzte Titel für einen gegebenen OAI-Identifier
`oai_subject`	liefert alle Schlagworte / Klassifikationen für einen gegebenen OAI-Identifier
`oai_creator`	liefert alle Autoren für einen gegebenen OAI-Identifier
`oai_date`	liefert das Veröffentlichungsdatum für einen gegebenen OAI-Identifier
`oai_type`	liefert den Dokumententyp für einen gegebenen OAI-Identifier
`oai_description`	liefert die Beschreibungen (Abstracts) für einen gegebenen OAI-Identifier
`oai_format`	liefert die Formate für einen gegebenen OAI-Identifier
`oai_language`	liefert die Sprache für einen gegebenen OAI-Identifier
`oai_contributor`	liefert die universitäre Einrichtung für einen gegebenen OAI-Identifier
`oai_publisher`	liefert den Veröffentlicher (Universitätsbibliothek) für einen gegebenen OAI-Identifier
`oai_rights`	liefert die rechtlichen Bedingungen für einen gegebenen OAI-Identifier
`oai_identifier`	liefert die absolute URL für einen gegebenen OAI-Identifier
`oai_relation`	liefert die Literaturquellen für einen gegebenen OAI-Identifier
`oai_identifier_exists`	testet, ob ein gegebener OAI-Identifier exisitiert
`oai_identifiers`	liefert alle absoluten URL's des Archivs in einem gegebenen Datumsbereich bzw. für einen gegebenen *Set*

Tabelle A59 *- Übersicht der gespeicherten Prozeduren (Teil 2)*

C.2. Schema

`config_hsss`

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
lastchanged	datetime	-	-	√	Datum der letzten Änderung
hsssdataxml	text	-	-	-	Die XML-basierte Konfigurationsstruktur. Der Wert `null` ist erlaubt, damit nicht von vornherein 2K belegt werden.

Tabelle A60 *- die Tabelle* `config_hsss`

`config_metadatasets`

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
kindofdocument	varchar(30)	-	-	√	Dokumentenart (Dissertation/Bericht)
lastchanged	datetime	-	-	√	Datum der letzten Änderung
metadatasetxml	text	-	-	-	Die XML-basierte Konfigurationsstruktur. Der Wert `null` ist erlaubt, damit nicht von vornherein 2K belegt werden.

Tabelle A61 *- die Tabelle* `config_metadatasets`

`stat_metadatasets`

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
processingid	char(100)	-	-	√	eindeutige Bearbeitungsnummer des Metadatensatzes
kindofdocument	varchar(30)	-	-	√	Dokumentenart (Dissertation/Bericht)
creationdate	datetime	-	-	√	Eingangsdatum des Metadatensatzes
finishingdate	datetime	-	-	√	Einstellungsdatum des Metadatensatzes
swbexportdate	datetime	-	-	-	Datum des Metadatenexports nach DDB
ddbexportdate	datetime	-	-	-	Datum des Metadatenexports nach SWB
swbfeedbackdate	datetime	-	-	-	Datum der OK-Rückmeldung DDB
ddbfeedbackdate	datetime	-	-	-	Datum der OK-Rückmeldung SWB
*unique{**processingid**}*					

Tabelle A62 *- die Tabelle* `stat_metadatasets`

`temp_metadatasets`

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
processingid	char(100)	-	-	√	eindeutige Bearbeitungsnummer des Metadatensatzes
kindofdocument	varchar(30)	-	-	√	Dokumentenart (Dissertation/Bericht)
creationdate	datetime	-	-	√	Eingangsdatum des Metadatensatzes
lastprocessingdate	datetime	-	-	√	Datum der letzten Bearbeitung
state	varchar(30)	-	-	√	Gesamtzustand des Metadatensatzes
lastprocessor	varchar(100)	-	-	√	letzter Bearbeiter
metadataset	text	-	-	-	der XML-basierte Metadatensatz (aktueller Stand)
metadatasetcopy	text	-	-	-	das XML-basierte Metadatensatz-Original des Autors
*unique{**processingid**}*					

Tabelle A63 *- die Tabelle* `temp_metadatasets`

`temp_metadatasets_finished`

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
processingid	char(100)	-	-	√	eindeutige Bearbeitungsnummer des Metadatensatzes
kindofdocument	varchar(30)	-	-	√	Dokumentenart (Dissertation/Bericht)
creationdate	datetime	-	-	√	Eingangsdatum des Metadatensatzes
lastprocessingdate	datetime	-	-	√	Datum der letzten Bearbeitung
state	varchar(30)	-	-	√	Gesamtzustand des Metadatensatzes
lastprocessor	varchar(100)	-	-	√	letzter Bearbeiter
metadataset	text	-	-	-	der XML-basierte Metadatensatz (aktueller Stand)
metadatasetcopy	text	-	-	-	der XML-basierte Metadatensatz (Stand des letzten Einstellens)
*unique{**processingid**}*					

Tabelle A64 *- die Tabelle* `stat_metadatasets_finished`

pers_address

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
place	varchar(100)	-	-	√	Ort
code	varchar(50)	-	-	-	Postleitzahl
street	varchar(100)	-	-	-	Straße
streetnumber	numeric(4)	-	-	-	Hausnummer
country	varchar(100)	-	-	-	Land
*unique{**code, country, place, street, streetnumber**}*					

Tabelle A65 *- die Tabelle* `pers_address`

pers_date

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
date	datetime	-	-	√	Datum
*unique{**date**}*					

Tabelle A66 *- die Tabelle* `pers_date`

pers_description

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
description	text	-	-	√	Beschreibung, Abstract
`language_id`	numeric(5)	-	-	√	die Sprache der Beschreibung (Fremdschlüssel)
*`language_id` references `pers_language`{**id**}*					

Tabelle A67 *- die Tabelle* `pers_description`

pers_format

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
extension	varchar(20)	-	-	√	Erweiterung des Dokumentenformats
mimetype	varchar(100)	-	-	√	Mime-Typ
name	varchar(100)	-	-	-	gebräuchliche Bezeichnung des Formats
description	varchar(255)	-	-	-	optionale Beschreibung
*unique{**extension**}*					

Tabelle A68 *- die Tabelle* `pers_format`

pers_kindofdocument

Name	Datentyp	I	P	M	Beschreibung
id	numeric(4)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(100)	-	-	√	Bezeichnung der Dokumentenart
description	varchar(255)	-	-	-	optionale Beschreibung

Tabelle A69 *- die Tabelle* ***pers_kindofdocument***

pers_language

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(100)	-	-	√	Name der Sprache
acronym	char(10)	-	-	√	Abkürzung der Sprache nach bestimmten Schema
scheme	varchar(50)	-	-	-	verwendetes Schema (ISO66..)
annotation	varchar(255)	-	-	-	optionale Bemerkung
*unique{**acronym**}*					

Tabelle A70 *- die Tabelle* ***pers_language***

pers_person

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
firstname	varchar(100)	-	-	√	Vorname
lastname	varchar(100)	-	-	√	Nachname
title	varchar(100)	-	-	-	Titel, Anrede, etc.
*unique{**firstname, lastname, title**}*					

Tabelle A71 *- die Tabelle* ***pers_person***

pers_publication

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(255)	-	-	√	Titel einer Publikation
isbn	varchar(100)	-	-	-	ISBN der Publikation
annotation	varchar(255)	-	-	-	optionale Bemerkung
*unique{**name**}*					

Tabelle A72 *- die Tabelle* ***pers_publication***

pers_relation

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
relation	varchar(255)	-	-	√	Literaturquelle (Name, URL, etc.)
annotation	varchar(255)	-	-	-	optionale Bemerkung
*unique{**relation**}*					

Tabelle A73 *- die Tabelle* `pers_relation`

pers_rights

Name	Datentyp	I	P	M	Beschreibung
id	numeric(4)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(255)	-	-	√	Name der Ressource für die rechlichen Bedingungen
version	varchar(100)	-	-	√	Version
url	varchar(255)	-	-	-	URL
*unique{**name, version**}*					

Tabelle A74 *- die Tabelle* `pers_rights`

pers_subject

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
subject	varchar(255)	-	-	√	Schlagwort, Klassifikation
*unique{**subject**}*					

Tabelle A75 *- die Tabelle* `pers_subject`

pers_contributor

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
university	varchar(100)	-	-	√	Universität
faculty	varchar(100)	-	-	√	Fakultät
department	varchar(100)	-	-	-	Fachbereich
institute	varchar(100)	-	-	√	Institut
`address_id`	numeric(6)	-	-	√	Adresse der Einrichtung (Fremdschlüssel)
`address_id` *references* `pers_address{id}`					
*unique{**university, faculty, department, institute**}*					

Tabelle A76 *- die Tabelle* `pers_contributor`

pers_publisher

Name	Datentyp	I	P	M	Beschreibung
id	numeric(4)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(255)	-	-	√	Name der Universitätsbibliothek bzw. des Veröffentlichers
address_id	numeric(6)	-	-	√	Adresse der Universitätsbibliothek / des Veröffentlichers (Fremdschlüssel)
*address_id references pers_address{**id**}*					
*unique{**name**}*					

***Tabelle A77** - die Tabelle pers_publisher*

pers_type

Name	Datentyp	I	P	M	Beschreibung
id	numeric(4)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
name	varchar(100)	-	-	√	Bezeichnung des Dokumententyps (innerhalb der Dokumentenart)
kindofdocument_id	numeric(4)	-	-	√	Dokumentenart (Fremdschlüssel)
*kindofdocument_id references pers_kindofdocument{**id**}*					
*unique{**name**}*					

***Tabelle A78** - die Tabelle pers_type*

pers_title

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
title	text	-	-	√	Titel
language_id	numeric(5)	-	-	√	Sprache des Titels (Fremdschlüssel)
*language_id references pers_language{**id**}*					

***Tabelle A79** - die Tabelle pers_title*

pers_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
processingid	char(20)	-	-	√	eindeutige Bearbeitungsnummer des Metadatensatzes
identifier_virtual	varchar(255)	-	-	√	absolute (virtuelle) URL
identifier_real	varchar(255)	-	-	√	reale URL
language_id	numeric(5)	-	-	√	Sprache der Hochschulschrift (Fremdschlüssel)
type_id	numeric(4)	-	-	√	Dokumententyp (Fremdschlüssel)
publisher_id	numeric(4)	-	-	√	Universitätsbibliothek / Veröffentlicher (Fremdschlüssel)
contributor_id	numeric(5)	-	-	√	universitäre EInrichtung (Fremdschlüssel)
rights_id	numeric(4)	-	-	√	rechtliche Bedingungen (Fremdschlüssel)

*language_id references pers_language{**id**}*
*type_id references pers_type{**id**}*
*publisher_id references pers_publisher{**id**}*
*contributor_id references pers_contributor{**id**}*
*rights_id references pers_rights{**id**}*
*unique{**processingid**}*

***Tabelle A80** - die Tabelle pers_metadataset*

pers_fulltext

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	Eindeutige Identifikationsnummer für die Tabelle
fulltext	text	-	-	-	Volltext der Hochschulschrift
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)

*metadataset_id references pers_metadataset{**id**}*
*unique{**metadataset_id**}*

***Tabelle A81** - die Tabelle pers_fulltext*

pers_date_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(6)	√	√	√	Eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Art des Datums (Abgabe, Veröffentlichung, etc.)
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
date_id	numeric(6)	-	-	√	Datum (Fremdschlüssel)

metadataset_id references ***pers_metadataset{id}***
date_id references ***pers_date{id}***
*unique{****metadataset_id, date_id****}*

Tabelle A82 *- die Tabelle* ***pers_date_metadataset***

pers_description_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	Eindeutige Identifikationsnummer für die Tabelle
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
description_id	numeric(6)	-	-	√	Beschreibung / Abstract (Fremdschlüssel)

metadataset_id references ***pers_metadataset{id}***
description_id references ***pers_description{id}***
*unique{****metadataset_id, description_id****}*

Tabelle A83 *- die Tabelle* ***pers_description_metadataset***

pers_format_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Typ des Formats (Abgabeformat, Originalformat, etc.)
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
format_id	numeric(6)	-	-	√	Format (Fremdschlüssel)

metadataset_id references ***pers_metadataset{id}***
format_id references ***pers_format{id}***
*unique{****metadataset_id, format_id****}*

Tabelle A84 *- die Tabelle* ***pers_format_metadataset***

pers_person_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Typ, Rolle der Person (Autor, Gutachter, etc.)
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
person_id	numeric(6)	-	-	√	Person (Fremdschlüssel)
*metadataset_id references pers_metadataset{**id**}*					
*person_id references pers_person{**id**}*					
*unique{**metadataset_id, person_id**}*					

***Tabelle A85** - die Tabelle pers_person_metadataset*

pers_publication_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
publication_id	numeric(6)	-	-	√	Publikation (Fremdschlüssel)
*metadataset_id references pers_metadataset{**id**}*					
*publication_id references pers_publication{**id**}*					
*unique{**metadataset_id, publication_id**}*					

***Tabelle A86** - die Tabelle pers_publication_metadataset*

pers_relation_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Typ der Relation (intern, extern)
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
relation_id	numeric(6)	-	-	√	Relation (Fremdschlüssel)
*metadataset_id references pers_metadataset{**id**}*					
*relation_id references pers_relation{**id**}*					
*unique{**metadataset_id, relation_id**}*					

***Tabelle A87** - die Tabelle pers_relation_metadataset*

pers_subject_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Typ des Subjects (Schlagwort (eigene, SWD); Klassifikation (DNB, RVK))
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
subject_id	numeric(6)	-	-	√	Subject (Fremdschlüssel)

*metadataset_id references pers_metadataset{**id**}*
*subject_id references pers_subject{**id**}*
*unique{**metadataset_id, subject_id**}*

***Tabelle A88** - die Tabelle pers_subject_metadataset*

pers_title_metadataset

Name	Datentyp	I	P	M	Beschreibung
id	numeric(5)	√	√	√	eindeutige Identifikationsnummer für die Tabelle
type	varchar(100)	-	-	√	Art des Titels (Originaltitel, alternativer Titel, übersetzter Titel)
metadataset_id	numeric(6)	-	-	√	Metadatensatz (Fremdschlüssel)
title_id	numeric(6)	-	-	√	Titel (Fremdschlüssel)

*metadataset_id references pers_metadataset{**id**}*
*title_id references pers_title{**id**}*
*unique{**metadataset_id, title_id**}*

***Tabelle A89** - die Tabelle pers_title_metadataset*

D. Verwendete Technologien / Produkte

Inhalt des Kapitels:

D.1. Kommerzielle Technologien / Produkte 276
D.1.1. Sybase 276
D.2. Open Source - Technologien / - Produkte 277
D.2.1. JSDK 277
D.2.2. Apache 277
D.2.3. Tomcat 278
D.2.4. Lucene 278
D.2.5. Xerces 279
D.2.6. Xalan 279
D.2.7. FOP 280
D.2.8. JDOM 280
D.2.9. PoolMan 281
D.2.10. PJ 281
D.2.11. FINJ 282
D.2.12. JavaMail 282

Dieser Abschnitt soll die in das Gesamtsystem integrierten Systeme, Programme bzw. API's jeweils kurz vorstellen. Dazu werden die Technologien in kommerzielle und Open-Source-Technologien unterschieden.

Der Fokus liegt dabei aber eher auf den auch zahlreich wesentlich stärker vertretenden Open-Source-Technologien, die jeweils insbesondere auch dahingehend vorgestellt werden, welche Funktion sie innerhalb der Gesamtanwendung erfüllen. So sind zukünftige Entwickler in der Lage, solche API's zu aktualisieren oder auch auszutauschen.

D.1. Kommerzielle Technologien / Produkte

D.1.1. Sybase

- **Produktname:**
  Sybase Adaptive Server Enterprise

- **Version:**
  12.0

- **Homepage:**
  `http://www.sybase.com/`

- **Beschreibung:**
  Bei dem Sybase Adaptive Server Enterprise (ASE) handelt es sich um ein relationales Datenbanksystem (RDBMS). Dabei unterstützt der ASE insbesondere Transaktions-intensive Anwendungen und Hochverfügbarkeitslösungen und wird daher oft in OLTP-Anwendungen verwendet.

- **Einsatzbereich innerhalb des Gesamtsystems**
  Der Datenbankserver wird ohne zusätzliche Module wie etwa Volltextsucherweiterung oder Java-Option betrieben und dient zur persistenten Speicherung der Metadatensätze und anderen Systeminformationen. Als JDBC Treiber dient der Sybase-eigene Treiber „JConnect“ in der Version 5.2.

D.2. Open Source - Technologien / - Produkte

D.2.1. JSDK

- **Produktname:**
  Java Standard Development Kit (JSDK)
- **Version:**
  1.3
- **Homepage:**
  `http://java.sun.com/`
- **Beschreibung:**
  Bei dem Java Standard Development Kit handelt es sich um eine komplette Programiersprachenumgebung mit allen notwendigen Werkzeugen (z. Bsp. Compiler, Interpreter etc.), Basisklassen und erweiterten API's (Bibliotheken). Alle Klassen dieses Systems sind auch im Quelltext verfügbar.
- **Einsatzbereich innerhalb des Gesamtsystems**
  Die gesamte Anwendung basiert auf den Klassen und Bibliotheken des JSDK in der Version 1.3 bzw. wurde mit dessen Hilfe realisiert.

D.2.2. Apache

- **Produktname:**
  Apache Webserver
- **Version:**
  1.3.14, gepatcht mid mod-ssl / openSSL
- **Homepage:**
  `http://www.apache.org/`
- **Beschreibung:**
  Der Apache Webserver ist der am weitesten verbreitete Webserver (etwa 60% Marktanteil) und gleichzeitig eines der größten und erfolgreichsten Open Source - Projekte.

- **Einsatzbereich innerhalb des Gesamtsystems**
  Der Apache Webserver dient zur Auslieferung von statischen Webseiten an die Clients (Browser). Für die Unterstützung von SSL wird „mod-sss“ verwendet, was auf Win32-Systemen aber nicht zufriedenstellend funktioniert. Hier sollte die Version 2.0 von Apache abgewartet werden.

D.2.3. Tomcat

- **Produktname:**
  Tomcat
- **Version:**
  3.1.2
- **Homepage:**
  `http://jakarta.apache.org/tomcat/index.html`
- **Beschreibung:**
  Tomcat ist ein Servlet-Container und gleichzeitig die Referenzimplementation für die von SUN spezifizierten Technologien Servlets und Java Server Pages (JSP). Tomcat ist wie Apache ein Open Source - Produkt und wird von der Apache Software Foundation (ASF) entwickelt.
- **Einsatzbereich innerhalb des Gesamtsystems**
  Der Webserver Apache interagiert über das Modul „mod_jk“, so dass Tomcat alle dynamischen Webinhalte serverseitig und javabasiert realisiert.

D.2.4. Lucene

- **Produktname:**
  Lucene
- **Version:**
  1.01b
- **Homepage:**
  `http://www.lucene.com/`

- **Beschreibung:**
  Bei Lucene handelt es sich um eine javabasierte Volltextsuchmaschine, die sich durch ein sauberes, objektorientiertes Design, eine flexible Indizierungsstrategie und durch eine gute Integrationsfähigkeit auszeichnet.
- **Einsatzbereich innerhalb des Gesamtsystems**
  Lucene wird im Rahmen des Gesamtsystems innerhalb der dritten Teilanwendung für die Volltextsuche und für die Feldsuche verwendet.

D.2.5. Xerces

- **Produktname:**
  Xerces Java Parser
- **Version:**
  1.3
- **Homepage:**
  `http://xml.apache.org/xerces-j/index.html`
- **Beschreibung:**
  Bei Xerces handelt es sich um einen javabasierten XML-Parser, der alle XML 1.0 - Empfehlungen des W3C [L-XL] unterstützt. Zusätzlich unterstützt dieser Parser die Industriestandards DOM Level 1 und SAX Version 1 sowie einige erweiterte Parser-Funktionalitäten wie XML Schema, DOM Level 2 und Sax Version 2. Xerces wird ebenfalls von der ASF entwickelt.
- **Einsatzbereich innerhalb des Gesamtsystems**
  Xerces wird für alle XML-Parsing-Vorgänge des Gesamtsystems in Verbindung mit JDOM verwendet, also insbesondere im Zusammenhang mit den XML-basierten Konfigurationsstrukturen.

D.2.6. Xalan

- **Produktname:**
  Xalan Java 2
- **Version:**
  2.0.1

- **Homepage:**
  `http://xml.apache.org/xalan-j/index.html`
- **Beschreibung:**
  Xalan ist ein XSLT-Prozessor, der es ermöglicht, XML-Dokumente in Text, HTML oder andere XML-Dokumententypen zu transformieren. Xalan implementiert die W3C-Empfehlungen für XSLT [L-XLa] und XPath und kann auf verschiedene Arten benutzt werden. Er wird ebenfalls von der ASF entwickelt.
- **Einsatzbereich innerhalb des Gesamtsystems**
  Xalan wird verwendet, um die XML-basierten Metadatensätze in andere Formate zu transformieren, z. Bsp. in Text (für E-Mails) oder XHTML (Exportformate DDB, SWB und Frontdoor).

D.2.7. FOP

- **Produktname:**
  FOP
- **Version:**
  0.19
- **Homepage:**
  `http://xml.apache.org/fop/index.html`
- **Beschreibung:**
  FOP ist der erste Druckformatierer, der auf XSL Formating Objects (FO) basiert. FOP kann aus einem XML-basierten Formating Object Tree PDF zur Laufzeit erzeugen und wird ebenfalls von der ASF entwickelt.
- **Einsatzbereich innerhalb des Gesamtsystems**
  FOP wird in Verbindung mit Xalan dazu verwendet, zur Laufzeit aus einem XML-basierten Metadatensatz ein PDF-Dokument zu generieren, das Teile dieses Metadatensatzes beschreibt.

D.2.8. JDOM

- **Produktname:**
  JDOM

- **Version:**
  1b7
- **Homepage:**
  `http://www.jdom.org/`
- **Beschreibung:**
  JDOM ist ein API für den javabasierten Zugriff, Manipulation und Erzeugung auf/von XML-Strukturen. Dabei nutzt JDOM intern Parser wie etwa Xerces, fungiert aber als Abstraktionsschicht und bietet Java-Entwicklern gewohnte Aufrufsemantiken und Datenstrukturen.
- **Einsatzbereich innerhalb des Gesamtsystems**
  JDOM wird in Verbindung mit Xerces an allen Stellen genutzt, an denen XML-Strukturen geparst bzw. erzeugt werden müssen.

D.2.9. PoolMan

- **Produktname:**
  PoolMan
- **Version:**
  1.4
- **Homepage:**
  `http://www.codestudio.com/`
- **Beschreibung:**
  PoolMan verbessert das Verhalten von datenbankbasierten Webanwendungen, indem es das Pooling und das Caching von Datenbankverbindungen ermöglicht.
- **Einsatzbereich innerhalb des Gesamtsystems**
  PoolMan wird bei fast allen Datenbankzugriffen verwendet, ist aber in der Konfigurationsdatei `web.xml` aktivierbar / deaktivierbar. Außerdem können Interna von PoolMan durch eine separate Konfigurationsdatei beeinflusst werden.

D.2.10. PJ

- **Produktname:**
  Etymon PJ

- **Version:**
  1.1
- **Homepage:**
  `http://www.etymon.com/pj/index.html`
- **Beschreibung:**
  PJ ist ein Entwicklungswerkzeug für das Parsen, Modifizieren und Erstellen von PDF-Dokumenten. Es ist dabei in der Lage, eine Objekt-Repräsentation eines vorhandenen PDF-Dokumentes im Hauptspeicher zu erstellen.
- **Einsatzbereich innerhalb des Gesamtsystems**
  PJ wird verwendet, um den Volltext aus PDF-Dokumenten zu extrahieren und an das PDF-Dokument eine Seite mit einem Zeitstempel des HSSS anzuhängen.

D.2.11. FINJ

- **Produktname:**
  FINJ
- **Version:**
  1.0.1
- **Homepage:**
  `http://finj.sourceforge.net/`
- **Beschreibung:**
  FINJ ist ein javabasiertes FTP-API und steht für FTP in Java. Es erlaubt den Zugriff auf FTP-Server nach RFC 959.
- **Einsatzbereich innerhalb des Gesamtsystems**
  FINJ wird innerhalb der *AdminTools* genutzt, um den FTP-basierten Metadatenexport nach SWB zu realisieren.

D.2.12. JavaMail

- **Produktname:**
  JavaMail

- **Version:**
  1.2
- **Homepage:**
  `http://java.sun.com/products/javamail/index.html`
- **Beschreibung:**
  Das JavaMail-API stellt ein plattformunabhängiges und protokollunabhängiges Framework zur Realisierung javabasierter Mail- und Messaginglösungen dar.
- **Einsatzbereich innerhalb des Gesamtsystems**
  JavaMail wird in der gesamten Anwendung zum Versenden von Mails verwendet, insbesondere in den *AdminTools* zur Realisierung des E-Mail-basierten Metadatenexports nach DDB.

E. Literaturverzeichnis und Linkverzeichnis

Literaturverzeichnis

[AAB00] Karl Avedal, Danny Ayers, Timothy Briggs and many others. «*Professional JSP*», Wrox Press, 2000

[Bec00] Kent Beck. «*Extreme Programming*», Addison-Wesley, 2000

[BeM00] Henning Behme, Steffen Mintert. «*XML in der Praxis*», Addison-Wesley, 2000, 2. Auflage

[BMR98] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stahl. «*Pattern-orientierte Softwarearchitektur*», Addison-Wesley, 1998

[Bog99] Marko Boger. «*Java in verteilten Systemen*», dpunkt, 1999

[Bra99] Jerry Bradenbaugh. «*JavaScript Application Cookbook*», O'Reilly, 1999

[BRJ99] Grady Booch, Jim Rumbaugh, Ivar Jacobson. «*Das UML-Benutzerhandbuch*», Addison-Wesley, 1999

[Che01] Su-Shing Chen. «*The Paradox of Digital Preservation*», Computer, March 2001

[Coo00] James W. Cooper. «*Java Design Patterns*», Addison-Wesley, 2000

[DDC99] James Duncan Davidson and Danny Coward. «*Java Servlet Specification - Version 2.2*», Sun Microsystems, Dec. 1999

[DeP00] Stefan Denninger und Ingo Peters. «*Enterprise JavaBeans*», Addison-Wesley, 2000

[DIN01] DINI Arbeitsgruppe "Elektronisches Publizieren". «*Empfehlungen zum Umgang mit elektronischen Hochschulpublikationen innerhalb von Universitäten, Bibliotheken und Verbünden*», Deutsche Initiative für Netzwerkinformationen (DINI), Stand Mai 2001

[DINa01] DINI Arbeitsgruppe "Elektronisches Publizieren", Dr. Bruno Klotz-Berendes. «*Empfehlung zu OAI-Setdefinitionen*[1]», Deutsche Initiative für Netzwerkinformationen (DINI), 18. Juni 2001 (DRAFT Status)

[Fla97] David Flanagan. «*JavaScript*», O'Reilly, 1997

[GHJ96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. «*Entwurfsmuster*», Addison-Wesley, 1996

[Gra98] Mark Grand. «*Patterns in Java: a catalog of reusable design patterns*», Wiley, 1998

[Gro00] Andreas Grote. «*Verflüchtigt - Der Zahn der Zeit nagt an den digitalen Daten*», c't (HEISE), 24/2000

[GuS00] Gunter Saake, Kai-Uwe Sattler. «*Datenbanken und Java*», dpunkt, 2000

[GVC00] Markus Gumbel, Marcus Vetter, Carlos Cárdenas. «*Java Standard Libraries: Java 2 Collection Framework und Generic Collection Library for Java*», Addison-Wesley, 2000

[Ham97] Graham Hamilton. «*JavaBeans Specification - Version 1.01*», Sun Microsystems, Jul. 1997

[Har00] Elliotte Rusty Harold. «*Java Network Programming*», O'Reilly, 2000, 2. Auflage

[HCF98] Graham Hammilton, Rick Catell, Maydene Fisher. «*JDBC: Datenbankzugriff mit JAVA*», Addison-Wesley, 1998

[HeS00] Andreas Heuer, Gunter Saake. «*Datenbanken: Konzepte und Sprachen*», mitp, 2000

[Hol00] Andreas Holubeck. «*Java Network Launcher Protocol und WebStart im Detail*», Java Magazin, 11-2000

[HuC98] Jason Hunter, William Crawford. «*Java Servlet Programming*», O'Reilly, 1998

[JMS01] Michael Jurisch, Christian Michel, Sebastian Schulz. «*Dokumentation der Konfigurationsdateien des HSSS*», SLUB, Jul. 2001

[LaL99] Ben Laurie and Peter Laurie. «*Apache*», O'Reilly, 1999

[1] Dieser Titel wurde vom Autor in Ermangelung eines bestehenden Titels vergeben. Leider liegt dieses Arbeitspapier auch nur in schriftlicher Form vor uns ist anderthalb Monate nach seiner Veröffentlichung noch nicht auf der Projektseite der Arbeitsgruppe verfügbar.

[Lea97] Doug Lea. «*Concurrent Programming in Java*», Addison-Wesley, 1997

[LFB00] Peter Lipp, Johannes Farmer, Dieter Bratko, Wolfgang Platzer, Andreas Sterbenz. «*Sicherheit und Kryptographie in Java*», Addison-Wesley, 2000

[McL00] Brett McLaughlin. «*Java and XML*», O'Reilly, 2000

[NaM99] Alexander Nakhimovsky and Tom Myers. «*Professional Java XML Programming with Servlets and JSP*», Wrox, 1999

[Nie99] Jennifer Niederst. «*Web Design in a Nutshell*», O'Reilly, 1999

[Oes98] Bernd Oestereich. «*Objektorintierte Softwareentwicklung*», Oldenbourg, 1998

[Pek00] Larne Pekowsky. «*JavaServer Pages*», Addison-Wesley, 2000

[PLC99] Eduardo Pelegrí-Llopart and Larry Cable. «*JavaServer Pages Specification - Version 1.1*», Sun Microsystems, Nov. 1999

[Rie97] Dirk Riehle. «*Muster für Entwurfswerkzeuge*», Addison-Wesley, 1997

[Roe00] Lars Röwekamp. «*Java TagLibs entwickeln und einbinden*», Java Magazin, 12-2000 und 01-2001

[RoS99] Peter Roßbach und Hendrik Schreiber. «*Java Server und Servlets*», Addison-Wesley, 1999

[Scha00] Jürgen Schmidt. «*Dasein oder Nicht-Dasein : Analyse der Ausfallzeiten von Webservern*», c't (HEISE), 8/2000

[Schb00] Sebastian Schulz. «*Webbasierte Übergabe von Hochschulschriften an die SLUB - ein Java-basierter Dokumentenserver für Hochschulschriften unter Nutzung der Servlet-/JSP-Technologie*», Großer Beleg im Studiengang Informatik, 2000

[Sie00a] Richard Sietmann. «*Von der Bibliothek ins Internet: Die wissenschaftlichen Bibliotheken stecken in einer Finanzkrise*», c't (HEISE), 26/2000

[Sie00b] Richard Sietmann. «*Elektronisches Publizieren - 'Public Domain' oder 'Private Property'?*», DMV-Mitteilungen, 1/2000

[SMG00] Matthias Schulz, Uwe Müller, Burghard Güther. «*Dokumentation der Dokumenttypdefinition "Dissertation Markup Language"*», Projekt "Digitale Dissertationen" der Humboldt-Universität zu Berlin, 2000

[SWB00] SWB-Arbeitspapier. «*Arbeitshilfe fuer die Bearbeitung von in den SWB eingespielten Metadaten"*», SWB-Arbeitspapier, Stand 31.10.2000

[TuP00] Volker Turau, Ronald Pfeiffer. «*Java Server Pages*», dpunkt, 2000

[Vli99] John Vlissides. «*Entwurfsmuster anwenden*», Addison-Wesley, 1999

[WiK99] Gerhard Wilhelms und Markus Kopp. «*Java professionell*», mitp, 1999

[Wil99] Alan R. Williamson. «*Java Servlets By Example*», Manning, 1999

[Wis01] Wissenschaftsrat. «*Empfehlungen zur digitalen Informationsversorgung durch Hochschulbibliotheken*», Wissenschaftsrat, 2001

[WLB97] Hakon Wium Lie and Bert Bos. «*Cascading Style Sheets*», Addison-Wesley, 1997

Linkverzeichnis

[L-AP] *Apache Software Foundation (ASF)*
`http://www.apache.org/`

[L-DB] *Dokumentenserver Der Deutschen Bbliothek*
`http://deposit.ddb.de/`

[L-DC] *Dublin Core*
`http://purl.org/DC/`

[L-DGf] *Deutsche Gesellschaft für Erziehungswissenschaft*
`http://www.dgfe.de/`

[L-DF] *Deutsche Forschungsgemeinschaft (DFG)*
`http://www.dfg.de/`

[L-DIa] *Deutsche Initiative für Netzwerkinformationen (DINI)*
`http://www.dini.de/`

[L-Di] *DiML2.0-DTD*
`http://dochost.rz.hu-berlin.de/epdiss/software/diml2_0.dtd`

[L-DIb] *DINI Arbeitsgruppe "Elektronisches Publizieren"*
`http://www.dini.de/workshops/epub/epub.php3`

[L-DOl] *Dissertationen Online*
`http://www.dissOnline.de/`

[L-IUK] *IuK-Initiative: Information und Kommunikation der wissenschaftlichen Fachgesellschaften in Deutschland*
`http://www.iuk-initiative.org/`

[L-JA] *Jakarta*
`http://jakarta.apache.org/`

[L-JB] *JavaBeans*
`http://java.sun.com/beans/`

[L-JD] *JDBC*
`http://java.sun.com/products/jdbc/index.html`

[L-JSa] *Java Servlet API*
`http://java.sun.com/products/servlet/`

[L-JSb] *Java Server Pages (JSP)*
`http://java.sun.com/products/jsp/`

[L-JW] *Java WebStart*
`http://www.javasoft.com/products/javawebstart/`

[L-LUa] *Lucene Search Engine*
`http://www.lucene.com`

[L-LUb] *Lucene FAQ*
`http://www.lucene.com/faq/`

[L-LUc] *Lucene Papers*
`http://www.lucene.com/papers/`

[L-LUd] *Lucene Talks*
`http://www.lucene.com/talks/inktomi/`

[L-MD] *METADISS Spezifikation V. 1.2*
`http://deposit.ddb.de/metadiss.htm`

[L-NC] *Netcraft Server Survey*
`http://www.netcraft.co.uk/survey/`

[L-OAa] *Open Archival Information System (OAIS)*
`http://ssdoo.gsfc.nasa.gov/nost/ref_model.html`

[L-OAb] *Open Archives Initiative (OAI)*
`http://www.openarchives.org`

[L-OAc] *OAI Protocol for Metadata Harvesting V. 1.1)*
`http://www.openarchives.org/OAI/openarchivesprotocol.htm`

[L-OAd] *OAI Registration)*
`http://www.openarchives.org/registration.htm`

[L-OP] *Online Publikationsverbund der Region Stuttgart (OPUS)*
`http://elib.uni-stuttgart.de/opus/`

[L-OR] *Oracle8i Database Server)*
`http://www.oracle.com/ip/index.html?content.html`

[L-SC] *XML Schema*
`http://www.w3.org/XML/Schema/`

[L-SH] *SELFHTML*
`http://www.teamone.de/selfhtml/selfhtml.htm`

[L-SO] *Simple Object Access Protocol (SOAP)*
`http://www.w3.org/TR/SOAP/`

[L-ST] *Struts Framework*
`http://jakarta.apache.org/struts/index.html`

[L-SW] *Serverwatch*
`http://serverwatch.internet.com/halloffame.html`

[L-SY] *Sybase Adaptive Server Enterprise (ASE)*
`http://www.sybase.com/products/databaseservers/ase`

[L-TC] *Tomcat*
`http://jakarta.apache.org/tomcat/`

[L-XL] *XML*
`http://www.w3.org/XML/`

[L-XLa] *XSL and XSLT*
`http://www.w3.org/Style/XSL/`